实例故事

Realy Taste Storys

笔者自 2000 年接触 Python 到现在，一直对 Python 的兴趣和信心有增无减。但 Python 在中国还处于推广普及的发展阶段。回想自己的学习体会，基本是从“不知己不知”到“不知己知”，再到“知己知”，最后“知己不知”。具体来说是开始涉及一个全新技术领域时，不知道这个领域的任何信息，连“不知道什么”都没有概念；后来，通过各种途径获得了部分相关信息，但不知道自个儿已经知道了哪些实用信息。再后来，通过实践切实掌握了领域的基础知识，明确在领域内掌握了什么；最后，掌握了领域间的关系，明白自己究竟还有多少知识不知道。感觉只有快速达到“知己知”的阶段，才可能事半功倍地继续学习下去。

我想把自己的这种学习体验与心得与大家分享，但怎么向那些没有摸到 Python 秉性的人们宣传这种体验呢？developerWorks 中 David Mertz 创作的“可爱的 Python”系列也就成了本书的原型结构。（访问地址：*http://www.ibm.com/developerworks/cn/linux/theme/special/index.html#python*，精巧地址：*http://bit.ly/mfUT*）

下面两个以解决实际问题为出发点，有剧情有人物的小白成长故事，将 Python 最可爱的方面，以小篇幅的轻松形式组织起来！希望读者可以跟随小白轻松体验到 Pythonic。

CDays “ 光盘故事 ”

CDay–5　Python 初体验和原始需求　3
CDay–4　可用的首个 Python 脚本　9
CDay–3　通过函式进行功能化　16
CDay–2　完成核心功能　22
CDay–1　实用化中文　31
CDay0　时刻准备着！发布　41
CDay+1　优化！对自个儿的反省　46
CDay+2　界面！不应该是难事儿　54
CDay+3　优化！多线程　69
CDayN　基于 Python 的无尽探索　75

CDay–5　Python 初体验和原始需求

Just use it! don't learn!——只用，不学!

剧本背景

本书采用实例故事的形式来讲解 Python，所谓实例故事，就是设计一个具体情景，让代表读者的初学者，同代表作者的行者沟通，从而完成学习过程，在过程中引导式地给读者展示 Python 的乐趣；当然读者不一定什么都不知道，作者也可能高明不到哪里去，但是，有个具体的事，讲起来就更有针对性一些。

好的，这就开始。依照传统说书的方式，先来个定场诗活跃一下气氛。

左咖啡，右宝石；还是灵蟒最贴心！

最贴心，不费心，用好只须听故事。

想清楚，就清楚，一切自己来动手！

要清爽，常重构！刚刚够用是王道！

下面正式开场。

人物介绍

书中会涉及两个人物，一个是小白，一个是行者，他们分别代表——

小白：

没有或是仅有一点编程体验的好奇宝宝，想快速上手使用 Python 解决实际问题。

行者：

啄木鸟/CPyUG 等中国活跃 Python 社区的热心 Python 用户，说话可能有些颠三倒四，但是绝对都是好心人。

约定

下面是本书所使用的一些体例约定。

列表：

指的是**邮件列表**——一种仅仅通过邮件进行群体异步交流的服务形式，是比 BBS 更加古老和有效的沟通方式。

小结：

在每日故事讲完之后会通过小结的方式，将当日故事情节中涉及的知识点和领域技术进行集中简述，以便读者明确要点。

练习：

每日故事的最后一节内容，虽然它所包含的问题和故事内容可能没有太大关联，但是这些问题必须使用前述涉及的知识点和领域技术才可以解决，所以特别列出，建议读者独立进行尝试，加强对相关知识的理解。

习题解答发布在图书维基：

http://wiki.woodpecker.org.cn/moin/ObpLovelyPython/LpyAttAnswerCdays

精巧地址：*http://bit.ly/XzYIX*

用 SVN 下载：

http://openbookproject.googlecode.com/svn/trunk/LovelyPython/exercise/part1-CDays/

邮件列表有古老的规范和格式。访问地址：
http://www.woodpecker.org.cn/share/classes/050730-CPUG/usMaillist/
精巧地址：
http://bit.ly/43WKcR
CPyUG 社区有丰富的列表资源。
访问地址：
http://wiki.woodpecker.org.cn/moin/CPUGres
精巧地址：
http://bit.ly/vrqUk

事件

小白忽然厌烦了不断地下载安装、破解，却总是找不到称心软件的生活：

“烦人！什么破软件这么不好使，还要 150$!!! 我!要!自个儿写!”

发动

究竟怎么回事儿呢？小白到列表中一说，大伙这才明白，原来他买了台刻录机，于是没日没夜地进行 eMule 的下载，才一个月刻录出来的光盘就有了上百张，结果，当他想找回一个专辑的 MP3 时，却遍寻不着……所以他想要一种工具，**不用插入光盘就可以搜索所有光盘的内容**。就这么简单的一个愿望，可是咋就找不到好用的软件呢?!这才有了上述那一幕。

Python!

OK！你们都说 Python 好用，我就来尝试一下吧！我是菜鸟我怕谁?!

运行环境：

推荐 `ActivePython`，虽然此乃商业产品，却是一个有自由软件版权保证的完善的 Python 开发应用环境，关键是文档及相关模块的预设都非常齐备。在 GNU/Linux 环境中，当然推荐使用原生的 Python.org，主流的自由操作系统发行版都内置了 Python 环境；对应的软件仓库中都有合适的 Python 版本可以选择，安装和使用也非常方便。

好了，下载、安装吧……

PCS3 “交互环境之 winpy”含有关于 ActivePython 的简介，网址为：http://www.activestate.com/Products/ActivePython/ ActiveState 是商业公司，但是对自由软件支持良多。Python.org 的网址为：http://www.python.org/download/ 它是 Python 语言本身的大本营。

Hello World!

“Hello World”非常非常著名，但凡是编程语言，第一课都要玩这个例程，下面我们也看一看 Python 的，如图 CDay–5-1 所示！

```
~/LovelyPython$ python
Python 2.5.2 (r252:60911, May  7 2008, 15:19:09)
[GCC 4.2.3 (Ubuntu 4.2.3-2ubuntu7)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello Pythonic World!"
Hello Pythonic World!
```

图 CDay–5-1　Hello World 示例

再展示一个类似的，但是推荐的体验环境为 iPython，如图 CDay–5-2 所示。

PCS2 “交互环境之 iPython”，相关网址为：http://ipython.scipy.org/ iPython 是个融合了 N 多 Unix Shell 环境特质的 Python 交互式命令行环境，推荐使用，你会爱上 Tab 键的。

```
~/LovelyPython$ ipython
Python 2.5.2 (r252:60911, May  7 2008, 15:19:09)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.4 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

In [1]: print "Hello Pythonic World!"
Hello Pythonic World!
```

图 CDay-5-2　Hello World 示例（iPython）

就是这么简单，告诉 Python 打印“Hello World!”就行了。

所以说，对于 Python，可以只用不学!

文档

丰富的文档可以安抚我们面对未知的恐惧，推荐深入阅读以下资料，但是不推荐现在就全面阅读。

Python Tutorial——Python 教程中文版本

在线访问：*http://wiki.woodpecker.org.cn/moin/March_Liu/PyTutorial*

精巧地址：*http://tinyurl.com/6h2q7g*

这是 CPyUG（Chinese Python User Group）中国 Python 用户组的资深专家刘鑫长期维护的一个基础文档，也是 Python 创造者 Guido.van.Rossum 唯一亲笔撰写的技术文档!

A Byte Of Python——简明 Python 教程

在线访问：*http://www.woodpecker.org.cn/share/doc/abyteofpython_cn/chinese/index.html*

精巧地址：*http://tinyurl.com/5k8pv5*

这是由沈洁元翻译的一篇流传甚广的学习 Python 的小书，从初学者的角度快速说明了一些关键知识点。原作者是印度的一位年轻的程序员，大家可以到这本书的网站直接和作者沟通：*http://www.swaroopch.com/byteofpython/*。

Python 标准库的中文版

在线访问：*http://www.woodpecker.org.cn/share/doc/Python/_html/ PythonStandardLib/*

精巧地址：*http://tinyurl.com/5pmvkn*

由“Python 江湖 QQ 群”全体成员共同完成，Python 2.0 内置所有标准模块的说明，是

初学者开发过程中必备的参考。

ASPN——Python Reference~Activestate 公司 Python 参考资料汇总

在线访问：*http://aspn.activestate.com/ASPN/Python/Reference/*

精巧地址：*http://tinyurl.com/58t3sl*

原始需求

安装好了 Python 环境，在行者的指点下又收集了一批资料的链接，下面小白想真正开始软件的创造了。但是，行者又告诫道：

“明晰你的问题，当问题真正得到定义时，问题已经解决了一半。因为，程序不过是将人的思想转述为机器可以理解的操作序列而已；对于寻求快速解决问题，而不是研究问题的小白和 Pythoner 们，精确、恰当地描述问题，就等于写好了程序框架，余下的不过是让程序可以运行罢了。”

于是小白根据直觉将软件需求细化了一下：

不用插入光盘就可以搜索所有光盘的内容，等于说……

要将光盘内容索引自动储存到硬盘上
要根据储存到硬盘上的光盘信息进行搜索

就这两点，也仅此两点的需求，可以如何以及怎样通过 Python 实现？小白和读者一同期待……

小结

作为开始，今天小白决定使用 Python 来解决光盘内容管理这一实际问题，安装了 python 环境，运行了“Hello World!”实例。

OK！轻松的开始，但是，你知道你同时也获得了免费的绝对强大的科学计算器吗？

练习

1. 计算今年是否是闰年。判断闰年条件，满足年份模 400 为 0，或者模 4 为 0 但模 100 不为 0。

2. 利用 Python 作为科学计算器的特性，熟悉 Python 中的常用运算符，并分别求出：

 1） 12*34+78–132/6

 2） （12*（34+78）–132）/6、（86/40）**5，并利用 `math` 模块进行数学计算，分别求出：

 145/23 的余数

 0.5 的 sin 和 cos 值（注意 sin 和 cos 中的参数是弧度制表示法）

 提示：可通过 `import math; help("math")`查看 math 帮助。

3. 找出 0~100 的所有素数。

CDay–4　可用的首个 Python 脚本

寻找吧！不要先想着创造——Python 是自足的。

现在的需求

小白根据曾经下载过的几个类似商业软件名称，给自个儿的软件起了个名字，叫“PyCDC”，即 Python 制作的光盘收集器（CD Collector），或者命令工具（CD Commander）。

再次确认当前的需求：

1. 将光盘内容索引储存为硬盘上的文本文件；
2. 可以根据储存到硬盘上的光盘信息进行搜索。

因为小白痛恨数据库（只能通过神奇的 SQL 语句才能调动数据库中的数据，太不直观，太不人性化了!），所以这第一个需求是将光盘信息读取为文本文件。

文件是系统的事儿

现在首要问题是“**如何读取指定光驱中的文件列表信息？**”

行者仅仅给了一条批示：**文件是系统的事儿**。

listdir()

系统 → 操作系统→ `operating system` → `os` 模块

呜乎！小白搜索了一大圈才弄明白什么叫“系统”，而且找到了相应的“模块”。

在《简明 Python 教程》的第 14 章“Python 标准库——os 模块”中看到一句话：os.listdir() 返回指定目录下的所有文件和目录名。

PCS200 “os(.stat; .path)”中对 os 模块有进一步细节描述。
《简明 Python 教程》的第 14 章“Python 标准库”的访问地址:http://www.woodpecker.org.cn/share/doc/abyteofpython_cn/chinese/ch14s03.html
精巧地址：http://bit.ly/2F1XXY

看起来可以利用这个函数，所以，创建第一个执行脚本 CDay-4-1.py：

```
# -*- coding: utf-8 -*-
import os
print os.listdir("/media/cdrom0")
```



代码注解：

1. `# -*- coding: utf-8 -*-`应该像八股文一样在每个脚本的头部声明，这是个忠告——为了解决中文兼容问题，同时你应该选择支持 Unicode 编码的编辑器环境，保证在运行脚本中的每个汉字都是使用 utf-8 编码过的。
2. `import os` 就是告诉 Python 环境，我们要使用 os 模块，依此类推，如果你想使用任何已有的模块包，就使用“import 模块名”的形式引用。
3. 最后一行才是我们真正想做的事儿：打印光盘根路径中的所有文件和目录。

PCS5 “Python 脚本文件”中进一步说明了脚本文档在各种操作系统中的表现，以及基础性的知识。

同时，小白也进一步了解了什么是脚本，为什么会有脚本。

因为 Python 程序不用编译，它是人写的程序文件，可以直接执行，就像话剧脚本一样，所以也称 Python 程序文件为脚本。在交互式命令行环境中，固然可以立即获得执行结果，但是在尝试过程中的各种程序代码无法被记忆下来，所以，事先写好脚本很重要，使用运行命令“python 脚本名.py”的形式就可以反复执行所有代码行。脚本是 Python 丰富的运行形式之一，也是最常用的一种。

上面的脚本运行结果类似图 CDay-4-1：

```
~/LovelyPython/CDays/cday-4$ python CDay-4-1.py
['README.diskdefines', 'install', 'preseed', 'pool', 'doc', 'pics', 'isolinux',
'md5sum.txt', 'ubuntu', '.disk', 'dists']
```

图 CDay-4-1 使用脚本运行

当然在 iPython 环境下，逐步运行也一样，如图 CDay-4-2 所示。

```
In [1]: import os

In [2]: print os.listdir("/media/cdrom0")
['README.diskdefines', 'install', 'preseed', 'pool', 'doc', 'pics', 'isolinux',
'md5sum.txt', 'ubuntu', '.disk', 'dists']
```

图 CDay-4-2 iPython 中运行

以上截屏输出是使用 Ubuntu 6.06 安装光盘来测试的结果，当然，小白使用进一步的目录

时，os.listdir()的确可以报告指定目录的所有信息，但是，问题是如何自动地将整个光盘中的所有文件和目录信息都“一次性地扫描”出来？

当然，自个儿写一个是可以的，不过这只是根据每级目录的信息再次不断调用os.listdir()，将所有层次的目录信息都逐一汇报出来而已，这样做值得吗？小白的热情是非常容易被看来很麻烦的代码实现击碎的……于是热心的无所不能的行者又提了句：“使用 walk()”。



提示：笔者使用 Ubuntu 系统，对于文件系统的使用和小白所使用的 Windows 系统稍有不同：Ubuntu 没有分区的概念，光盘一般会自动统一地挂接到 /media/cdrom0 目录，而小白则可以自由地使用 G: \等类似的盘符进行替换。放心，Python 足够聪明，它是跨平台的。须要提醒的是，Windows 中最好使用 d: \\进行分区的指定。否则，会出现类似 d: /tmp\something 的混合输出。

walk

输出目录下整个信息，包括子目录

一阵乱搜后，发现居然有个单独的模块 os.path 是进行文件路径处理的!

再一搜索英文，发现：

```
Note: The newer os.walk() generator supplies similar functionality and can be easier to use.
```

好像在我们的环境中有两个 walk()：os.path.walk()和 os.walk()，是不是后一个更新，也更好用?

按照 os-file-dir 的示例改造脚本，尝试 CDay-4-2.py：

```
1 # -*- coding: utf-8 -*-
2 import os
3 for root, dirs, files in os.walk('/media/cdrom0'):
4   print root,dirs,files
```

然后，按照脚本运行方式尝试：~$ python CDay-4-2.py... 哇！一下子输出了一大堆的东西！看来管用，应该是将整个光盘的文件/目录信息都扫描出来了。

PCS2 PCS2 “交互环境之 iPython”详细说明了加强的交互命令行界面 iPython 的甜美；简要地说 iPython 是 Python 原生交互环境的加强，追加了很多纯正 Unix 风味的响应特性。

Ubuntu 是最流行的 GNU/Linux 发行版。
官方网站：
http://www.ubuntu.com/
中文官方网站：
http://www.ubuntu.org.cn/

PSC200 PSC200 “os(.stat; path)”含有 os 模块进一步细节，其中 os.path()函式的详细说明可在线查阅。
访问地址：
http://aspn.activestate.com/ASPN/docs/ActivePython/2.4/python/lib/module-os.path.html
精巧地址：
http://bit.ly/2XD0k0

PCS4 PCS4 “常用自省”中详细介绍了最常用的几个 Python 内建函式，它们可以通过自省的方式来快速对 Python 进行查询，寻找想要的处理支持。

不管三七二十一，先保存为文件！

PCS200 “os(.stat; .path)”中有 os 模块的进一步细节描述，其中 os-file-dir 可以在线查阅：
http://aspn.activestate.com/ASPN/docs/ActivePython/2.4/python/lib/os-file-dir.html
精巧地址：
http://bit.ly/eJj4f

输出成文件

小白根据行者的指引，同时也进行了搜索，了解了 file_objects 文件对象操作相关知识，然后自己加了点儿想象就组成了以下脚本 CDay-4-3.py：

```
1 # -*- coding: utf-8 -*-
2 import os
3 for root, dirs, files in os.walk('/media/cdrom0'):
4   open('mycd.cdc', 'a').write(root+dirs+files)
```

尝试执行~$python CDay-4-3.py，如图 CDay-4-3 所示。

```
~/LovelyPython/CDays/cday-4$ python CDay-4-3.py
Traceback (most recent call last):
  File "CDay-4-3.py", line 4, in <module>
    open('mycd.cdc', 'a').write(root+dirs+files)
TypeError: cannot concatenate 'str' and 'list' objects
```

图 CDay-4-3　初步 CDay-4-3.py 脚本运行结果

PCS102 “For 循环”进一步说明了小白首次遇见的新语法现象；这也是一切程序中最基本的反复运行相似语句的技巧。

PCS200 “os(.stat; .path)”中对 os 模块有进一步的细节描述，其中 file_objects 可以在线查阅：
http://www.woodpecker.org.cn/diveintopython/file_handling/file_objects.html
精巧地址：
http://bit.ly/1e41SW

完了!报错!……小白感觉信心又降到了冰点。“呵呵”，行者说，“不用担心，在 Python 中想要解决 bug 是非常轻松的，因为它有完善的回溯能力。Python 可以非常精确地汇报小白出错的时间、地点、原因，我们只要对应修订就行了。”

现在的问题是：

```
TypeError: cannot concatenate 'str' and 'list' objects
```

也就是说，不能将 str 和 list 对象进行连接。这是非常直白的数据类型错误。

什么是“数据类型”？这问题可以复杂地解释，也可以简单地说。复杂的解释并不能帮助小白理解和记忆，简单的说也不一定能加深印象。通俗地讲呢，就是计算机内存空间中有不同的种族，好比鸡鸭同笼，长的都挺像，但是没有办法卖相同的价钱，除非都剁碎了炸成春卷——也就是说只要发生类似数据类型时，把它们转换成相同的类型就得了!

这种方式对于其他语言来说可能会很麻烦，但是在 Python 中是再简单不过的了。因为，**Python 里一切都是对象**。

“Python 里一切都是对象”是理解应用 Python 的关键之一，详细内容请参考《万物皆对象》：
http://www.woodpecker.org.cn/diveintopython/getting_to_know_python/everything_is_an_object.html
精巧地址：
http://bit.ly/2tiwrd

讨教行者后，他们给了个样例：

```
print "%s %s %s %s" % ("字串",["数","组"],("元","组"),{'字典': 123})
```

如果小白有 C 编程经验立即就可以联想到这是输出格式化的技巧，使用%s 的格式化声明，要求后面的对象以 String 字串的格式进行转换，所以脚本 CDay-4-4.py 的改造也就顺理

成章了 ：

```
1 # -*- coding: utf-8 -*-
2 import os
3 for root, dirs, files in os.walk('/media/cdrom0'):
4   open('mycd.cdc', 'a').write("%s %s %s" % (root,dirs,files))
```

代码注解：

1. 声明是 utf-8 编码文本；
2. 引入了 os 模块；
3. 使用 `os.walk()` 扫描光盘，并返回三个对象；
4. 使用 `open()`打开 mycd.cdc 文件对象，并声明成追加模式，逐行记录以上三个对象。

不过，执行没有报错，而且文件也生成了，但是为什么打不开?……，呵呵，那是另一个问题了。

小结

通过指点，以及在文档中狂乱的搜寻，小白今天顺利地将光盘信息全部扫描出来了，并存储成了文件!

不过仅仅 4 行代码，真正运行可用的也就两行代码，居然已经可以达到想要的软件功能的 50%……Python 是不是很神奇？

今天小白其实已经接触到了一大批概念：

1. 模块 —— `import os`;
2. 内置函式 —— `open()`;
3. 循环 —— `for ... in ...`;
4. 块界定符;
5. 注释符;
6. 对象;
7. 文件对象;
8. 对象转换;
9. 格式化声明 —— `%s`;
10. 数据类型。

PCS

涉及知识点的进一步信息请查阅对应的作弊条：

PCS100“ import ”说明引入模块的技巧

PCS205“ 内建函式 ”介绍一些最常用的内建函式

PCS102“ For 循环 ”介绍最常用的循环体

PCS103“ 缩进 ”阐述了 Python 最大的特点：使用缩进来区分语法单位

PCS104“ 注释 ”介绍了 Python 式的注释方式

PCS105“ 对象 ”介绍了 Python 中的一等公民

PCS106“ 文件对象 ”介绍了最常用的对外交互渠道：文件，及其处置对象

PCS107“ 字串格式化 ”介绍了 Python 内置的字符模板支持特性

PCS101“ 内建数据类型 ”介绍了动态语言的最大便利：自由类型转换的体验

不过它们都包含在了 4 行代码中。我们的原则是：先用后学，快速获得体验，然后寻求理论支持，所以，先不求甚解，达到目的，然后就自在了。

练习

1. os 模块中还有哪些功能可以使用？提示：使用 `dir()`和 `help()`。
2. `open()` 还有哪些模式可以使用?
3. 尝试 `for .. in ..`循环可以对哪些数据类型进行操作?
4. 格式化声明，还有哪些格式可以进行约定?
5. 下面的写入文件模式好吗？有改进的余地吗?

下面是 CDay–4-5.py ，它好在哪里?

```
1 # -*- coding: utf-8 -*-
2 import os
3 export = ""
4 for root, dirs, files in os.walk('/media/cdrom0'):
5   export+="\n %s;%s;%s" % (root,dirs,files)
6 open('mycd2.cdc', 'w').write(export)
```

以下的 CDay–4-6.py 又更加好在哪里?

```
1 # -*- coding: utf-8 -*-
2 import os
3 export = []
4 for root, dirs, files in os.walk('/media/cdrom0'):
5    export.append("\n %s;%s;%s" % (root,dirs,files))
6 open('mycd2.cdc', 'w').write(''.join(export))
```

6. 读取文件 cdays–4-test.txt 内容，去除空行和注释行后，以行为单位进行排序，并将结果输出为 cdays–4-result.txt。

```
#some words
Sometimes in life,
You find a special friend;
Someone who changes your life just by being part of it.
Someone who makes you laugh until you can't stop;
Someone who makes you believe that there really is good in the world.
Someone who convinces you that there really is an unlocked door just waiting
for you to open it.

This is Forever Friendship.
```

```
when you're down,
and the world seems dark and empty,
Your forever friend lifts you up in spirits and makes that dark and empty
world
suddenly seem bright and full.
Your forever friend gets you through the hard times,the sad times,and the
confused times.
If you turn and walk away,
Your forever friend follows,
If you lose you way,
Your forever friend guides you and cheers you on.
Your forever friend holds your hand and tells you that everything is going
to be okay.
```

CDay–3　通过函式进行功能化

不断否定自己，但要坚持最初的意愿——不论战术上如何变化，千万不要忘记战略目标。

如果小白真正理解和可以自如应用前一日所讲的 4 行代码中包含的各种知识，那么，离完成软件之日就已经不远了。

需求

首先小白根据已有的体验，对 PyCDC 的软件需求进行了进一步完善。

1. 将光盘内容索引存储为硬盘上的文本文件。
 1）存储成`*.cdc` 的文本文件；
 2）可以快速指定文件名。
2. 根据储存到硬盘上的光盘信息进行搜索。
 1）可以搜索指定目录中所有`*.cdc` 文件。

这样一来，可以看出 PyCDC 的使用其实分为两部分。

1. 刻录光盘时，将光盘信息通过 PyCDC 存储为对应光盘标号的`*.cdc` 文件。
2. 使用光盘时，在 PyCDC 中搜索，确认`.cdc` 文件名，即光盘标号，从而针对性地读取确切的光盘，不用遍寻所有光盘了!

功能化

简单地讲就是将以往验证想法的代码，变成可以方便使用的功能，让它可以重复在不同应用环境中使用。小白想象着自个儿的 PyCDC 可以像普通的命令行工具一样来使用：

```
python pycdc.py -e mycd1-1.cdc
#将光盘内容记录为 mycd1-1.cdc
python pycdc.py -e cdc/mycd1-1.cdc
#将光盘内容记录到 cdc 目录中的 mycd1-1.cdc
python pycdc.py -d cdc -k 中国火
#搜索 cdc 目录中的光盘信息，寻找有“中国火”字样的文件或是目录，在那张光盘中
可能还有其他的功能，但是最核心的功能应该就是这 3 样。
```

要想达到这种效果，最直接的的方法就是函式化!

函式化

声明函式名，定义参数，然后使用缩进，将前一日摸索出来的代码包装一下，使用参数代替原先指定的目录和文件名，请看 CDay−3-1.py。

```
1 # -*- coding: utf-8 -*-
2 import os
3 def cdWalker(cdrom,cdcfile):
4    export = ""
5    for root, dirs, files in os.walk(cdrom):
6       export+="\n %s;%s;%s" % (root,dirs,files)
7    open(cdcfile, 'w').write(export)
8 cdWalker('/media/cdrom0','cd1.cdc')
9 cdWalker('/media/cdrom0','cd2.cdc')
```

小白获得了第一个 Python 函式，并成功运行了两次，即将同张光盘的内容记录到不同的文件中！非常 easy。

PCS108

PCS108“ 函式 ”进一步说明了什么是函式，什么时候应该将代码集成为函式等知识。在 Python 中对象是一等公民，函式则是实际可用脚本中最基础的人民了。
精巧地址：
http://bit.ly/vrqUk

交互参数

但是如何从命令行获取输入的参数呢？

搜索或是询问后，从行者那儿又获得一个提示：`print sys.argv`。

那么，无畏的小白立即创建了一个将真正使用的功能脚本 pycdc-v0.1.py，并尝试加入了最新的提示代码：

```
1 # -*- coding: utf-8 -*-
2 import os
3 print sys.argv
4 def cdWalker(cdrom,cdcfile):
5     export = ""
6     for root, dirs, files in os.walk(cdrom):
7         export+="\n %s;%s;%s" % (root,dirs,files)
8     open(cdcfile, 'w').write(export)
9 #cdWalker('/media/cdrom0','cd1.cdc')
```

运行结果如图 CDay–3-1 所示：

```
~/LovelyPython/CDays/cday-3$ python pycdc-v0.1.py -e mycd-1.cdc
Traceback (most recent call last):
  File "pycdc-v0.1.py", line 6, in <module>
    print sys.argv
NameError: name 'sys' is not defined
```

图 CDay–3-1　pycdc-v0.1.py 运行结果

OK，出错了，但是有了 os 模块的经验，瞧着 sys.argv 这么眼熟，小白猜这是个模块，需要引入，所以微小地改动了一下代码：

```
1 # -*- coding: utf-8 -*-
2 import os,sys
3 print sys.argv
4 def cdWalker(cdrom,cdcfile):
5     export = ""
6     for root, dirs, files in os.walk(cdrom):
7         export+="\n %s;%s;%s" % (root,dirs,files)
8     open(cdcfile, 'w').write(export)
9 #cdWalker('/media/cdrom0','cd1.cdc')
```

运行结果如图 CDay–3-2 所求：

```
~/LovelyPython/CDays/cday-3$ python pycdc-v0.1.py -e mycd-1.cdc
['pycdc-v0.1.py', '-e', 'mycd-1.cdc']
```

图 CDay–3-2　修改 pycdc-v0.1.py 后的运行结果

PCS113 “交互参数”进一步分享了有关交互参数的使用方法，读者可以深入了解 sys 及其使用方法。

Great! 一切如愿，然后就是简单的识别参数问题了。

逻辑判断

好了，小白有了自个儿的体验：当面对不熟悉的工具时，会对应该输入什么参数一头雾水，此时，若软件能友好智能地进行提示，就太好了……

要做到这一点，得对未知用户的行为进行判定，对非期待的输入进行提示，而不是由 Python 自个儿出错中断。与其他语言类似，Python 中也有 if、else 等类似逻辑判别语句，不妨找到相关内容，照猫画虎：

```
# -*- coding: utf-8 -*-
import os,sys
CDROM = '/media/cdrom0'
def cdWalker(cdrom,cdcfile):
    export = ""
    for root, dirs, files in os.walk(cdrom):
        export+="\n %s;%s;%s" % (root,dirs,files)
    open(cdcfile, 'w').write(export)
#cdWalker('/media/cdrom0','cd1.cdc')
if "-e"==sys.argv[1]:
    cdWalker(CDROM,sys.argv[2])
    print "记录光盘信息到 %s" % sys.argv[2]
else:
    print '''PyCDC 使用方式:
    python pycdc.py -e mycd1-1.cdc
    #将光盘内容记录为 mycd1-1.cdc
    '''
```

代码注解：

1. 使用全局参数 `CDROM` 指定当前的光盘访问路径，总是`/media/cdrom0`；
2. 通过 `if "-e"==sys.argv[1]` 判定第二个参数是`-e` 时，使用第三个参数作输出文件名记录光盘信息，并输出提示；
3. 通过 `else` 捕获所有意外情况，输出错误提示，结束脚本。

PCS109 “系统参数”进一步分享了有关函式参数的使用技巧，对于 cdWalker(CDROM，sys.argv[2])，读者可以深入了解其使用方法。

PCS110 “逻辑分支”进一步详细说明了进行条件判别时 Python 支持的方式。

OK，看起来应该已经完成了一两项功能，但是实际运行时，结果如图 CDay–3-3 所示：

```
~/LovelyPython/CDays/cday-3$ python pycdc-v0.1.py -e cdc/mycd-1.cdc
Traceback (most recent call last):
  File "pycdc-v0.1.py", line 11, in <module>
    cdWalker(CDROM,sys.argv[2])
  File "pycdc-v0.1.py", line 8, in cdWalker
    open(cdcfile, 'w').write(export)
IOError: [Errno 2] No such file or directory: 'cdc/mycd-1.cdc'
```

图 CDay–3-3　pycdc-v0.1.py 运行 IO 错误

尝试带目录的输出时：

```
    IOError: [Errno 2] No such file or directory: 'cdc/mycd-1.cdc'
```

I/O——输入/出问题，当然啦，`cdc` 目录并不存在。

手工建立 cdc 目录后，再运行就 OK 了，所以功能设计细化为：

```
~$python pycdc.py -e mycd1-1.cdc
##第 2 个参数是"-e";使用 cdWalker()将光盘内容记录为 mycd1-1.cdc
~$python pycdc.py -e cdc/mycd1-1.cdc
##第 2 个参数是"-e";而且第 3 个参数中含有目录指定;
##就将光盘内容记录到 cdc 目录中的 mycd1-1.cdc;如果 cdc 目录不存在,可以自动创建。
~$python pycdc.py -d cdc -k 中国火
##第 2 个参数是"-d";而且第 4 个参数是"-k";
##就搜索 cdc 目录中的光盘信息，输出含有关键字“中国火”的文件或是目录，指出它在哪张光盘中。
```

完成以上所有功能和判定部分的伪代码：

```
if "-e"==sys.argv[1]:
    #判别 sys.argv[2]中是否有目录，以便进行自动创建
    cdWalker(CDROM,sys.argv[2])
    print "记录光盘信息到 %s" % sys.argv[2]
elif "-d"==sys.argv[1]:
    if "-k"==sys.argv[3]:
        #进行文件搜索
    else:
        print '''PyCDC 使用方式:
        python pycdc.py -d cdc -k 中国火
        #搜索 cdc 目录中的光盘信息，寻找有“中国火”字样的文件或是目录在哪张光盘中
        '''
else:
    print '''PyCDC 使用方式:
        python pycdc.py -d cdc -k 中国火
        #搜索 cdc 目录中的光盘信息，寻找有“中国火”字样的文件或是目录在哪张光盘中
        '''
```

到这一步，小白已经开始头大了，如果功能继续追加，软件进一步友好地自动识别各种意外情况，这棵逻辑树将会可怕地增长下去……

小白没有耐心和信心面对如此永无止境的意外情况的判别，于是开始寻找其他解决方案……

小结

为了快速将已知技巧转化为可用脚本，小白接触到了以下知识：

1. sys 模块的接收交互参数；
2. 函式概念；

3. 逻辑判定语句；
4. 伪代码技术，进行开发复杂度的估算。

PCS200 “os(.stat;.path)”详细说明了 os 及其包含模块的常用函式;分享了 os 在常见场景中的使用技巧。

练习

1. 根据 DiPy 10.6. 处理命令行参数（*http://www.woodpecker.org.cn/diveintopython/scripts_and_streams/command_line_arguments.html*，精巧地址：*http://bit.ly/1x5gMw*），使用 getopt.getopt()优化当前功能函式。
2. 读取某一简单索引文件 cdays−3-test.txt，其每行格式为：文档序号关键词，现须根据这些信息将它转化为倒排索引，即统计关键词在哪些文档中，格式如下：包含该关键词的文档数 关键词 => 文档序号。其中，原索引文件作为命令行参数传入主程序，并设计一个 collect 函式统计“关键字↔序号”结果对，最后在主程序中输出结果至屏幕。

 cdays−3-test.txt 内容：

```
1 key1

2 key2

3 key1

7 key3

8 key2

10 key1

14 key2

19 key4

20 key1

30 key3
```

3. 在一个 8×8 国际象棋盘上，有 8 个皇后，每个皇后占一格；要求皇后间不会出现相互“攻击”的现象，即不能有两个皇后处在同一行、同一列或同一对角线上，问共有多少种不同的方法。

CDay–2　完成核心功能

利用文本文件完成核心功能

没有完美的软件，够用并且容易使用的软件已经算是完美的了。

回顾需求

已经获得命令行工具样的程序的小白几乎可以脱离小白的称号了，他可以独立根据已往经验，不断尝试出成果来，就该是大白了。

现在可以将最初的需求细化成这样：

1. 将光盘内容索引存储为硬盘上的文本文件
 1） 存储成 *.cdc 的文本文件
 2） 可以快速指定文件名
 - 通过命令行调用 `python pycdc.py -e mycd1-1.cdc`
2. 根据储存到硬盘上的光盘信息进行搜索
 1） 可以搜索指定目录中所有*.cdc 文件
 - 通过命令行调用 `python pycdc.py -d cdc -k 中国火`

但是，要完成一个个看似简单，实际有 *N* 多情况的逻辑判定，实在是件令人沮丧的事儿，小白吼了声：“俺就是想做个简单的命令行工具，咋就这么难呢？”

热心的行者，又出声了：“使用 cmd 吧！”

cmd 模块

```
cmd ： Support for line-oriented command interpreters
```

咦？它是专门支持命令行界面的模块？！但是几乎没有可以照抄的代码呀！

照猫画虎的重要资源

不搜不知道，一搜吓一跳！原来英文技术书已在出版社网站上公开了书中的代码样例，比如：《Learning Python》（见图 CDay–2-1）、《Learning Python，Second Edition》。

按照大学里论文的制造模式进行相关尝试（即从已被证实正确的基础上加以改造），成功的机会比较大!

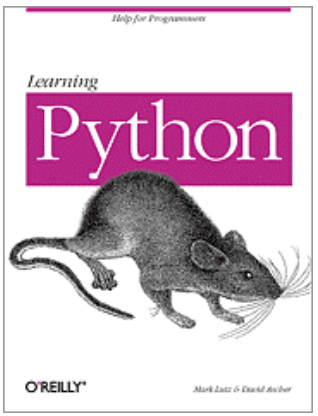


图 CDay–2-1　Learning Python

那么，我们可以参考第一版《Leaning Python》第 9 章的例程 rolo.py。

“资源索引”章节中有完整的在线中文文档的资源介绍，有意的读者可以在网络中针对性的对有兴趣的 Python 内容进一步学习。精巧地址：http://bit.ly/vrqUk

重构当前代码

重构（refactoring），听上去好像是很高级的编程技术，其实说穿了就是自个儿看不过去自个儿，于是不断地将代码修改得更加合理，使之运行起来更加高效。小白的目标是快速实现心中的功能，但是现在面对的又一个全新的模块，他想使用干净的脚本进行尝试，所以考虑重构，于是将原先的代码整理到 cdctools.py，创建新脚本 pycdc-v0.4.py：

```
1 # -*- coding: utf-8 -*-
2 import sys, cmd
```

```
 3 class PyCDC(cmd.Cmd):
 4     def __init__(self):
 5         cmd.Cmd.__init__(self)             # initialize the base class
 6     def help_EOF(self):
 7         print "退出程序 Quits the program"
 8     def do_EOF(self, line):
 9         sys.exit()
10
11     def help_walk(self):
12         print "扫描光盘内容 walk cd and export into *.cdc"
13     def do_walk(self, filename):
14         if filename == "":filename = raw_input("输入 cdc 文件名:: ")
15         print "扫描光盘内容保存到:'%s'" % filename
16
17     def help_dir(self):
18         print "指定保存/搜索目录"
19     def do_dir(self, pathname):
20         if pathname == "": pathname = raw_input("输入指定保存/搜索目录: ")
21
22     def help_find(self):
23         print "搜索关键词"
24     def do_find(self, keyword):
25         if keyword == "": keyword = raw_input("输入搜索关键字: ")
26         print "搜索关键词:'%s'" % keyword
27 if __name__ == '__main__':     # this way the module can be
28     cdc = PyCDC()          # imported by other programs as well
29     cdc.cmdloop()
```

这个 90% 照抄过来的脚本纯粹是用来尝试 `cmd` 模块功能的，没有任何实际作用，只是想通过打印输出信息，来印证自个儿的想法……

PCS404

PCS404 “代码重构浅说”中说明重构（Refactoring）是现代软件开发工程中常用的技巧，可以保证系统在可用状态下持续不断地提高代码品质！在 PCS404 中，浅显地说明了一下重构思想及常见的重构技巧。

猜测和学习

如何针对已证实有用的代码进行改造和学习?

猜测是最自然的方法，不用想着要学习什么浑厚的背景知识，只要按照自个儿的理解，给当前可用的代码一个说法，使之支持自个儿的自由发挥就对了。

运行一下，其结果如图 CDay−2-2 所示：

```
~/LovelyPython/CDays/cday-2$ python pycdc-v0.4.py
(Cmd) help

Documented commands (type help <topic>):
========================================
EOF  dir  find  walk

Undocumented commands:
======================
help

(Cmd) dir
输入指定保存/搜索目录: cdc
(Cmd) xx
*** Unknown syntax: xx
(Cmd) ? find
搜索关键词
(Cmd)
```

图 CDay–2-2　pycdc-v0.4.py 运行结果

经过几次尝试可以确认以下几点。

1. `if __name__ == '__main__'`：是进行自测运行用的，具体怎么回事儿先不管，好用就成。
2. `class PyCDC(cmd.Cmd)`：是进行类定义的，然后可以使用 `cdc = PyCDC()` 进行实体化，并使用类的各种功能。
3. `def __init__(self)`：是类实体化时自动运行的初始化函式。
4. `cmd.Cmd.__init__(self)`是初始化基类？不明白，反正要使用 `cmd` 模块，就这么来好了。
5. `def help_*()` 和 `def do_*()`应该成对出现，一个是打印功能的帮助信息，一个是实际干事儿的。
6. 利用 `cmd` 创建的应用可以自动处理各种意外情况，汇报`*** Unknown syntax: xxx`。

非常整齐对称的代码呀，小白不知道怎么可以不破坏代码的这种整齐，又可以加入原先的功能代码……

警告：运行时总是汇报的一个 Non-ASCII character '\xe6' ...警告，这是个中文问题，只要不影响我们的尝试，先不理会它，技术和知识是无限的，我们要先关注自个儿的目标，适当地放弃，有利于提高开发效率。

利用自个儿的模块

那么，如何直接利用暂存到 cdctools.py 的功能呢？

```
1 # -*- coding: utf-8 -*-
2 import os
3 def cdWalker(cdrom,cdcfile):
```

```
    export = ""
    for root, dirs, files in os.walk(cdrom):
        export+="\n %s;%s;%s" % (root,dirs,files)
    open(cdcfile, 'w').write(export)
if __name__ == '__main__':      # this way the module can be
    CDROM = '/media/cdrom0'
    cdWalker(CDROM,"cdc/cdctools.cdc")
```

询问行者得到的提示是：

```
from 某脚本文件名 import *
```

立即拿来改造 pycdc-v0.4.py：

```
# -*- coding: utf-8 -*-
import sys, cmd
from cdctools import *
...
if __name__ == '__main__':      # this way the module can be
    cdc = PyCDC()           # imported by other programs as well
    cdc.cmdloop()
    CDROM = '/media/cdrom0'
    cdWalker(CDROM,"cdc/cdctools.cdc")
```

PCS108“函式”、PCS105“对象”、PCS110“逻辑分支”进一步说明在不同级别的 Python 执行单位中，有哪些基础技巧。

哈哈哈!可以运行！而且悟出来一点，在代码中，按代码的复用尺度来分，从小到大应该是：

代码行→函式→类→模块

好像还有更大的一级**包**，具体现在还用不上，就先不管它了。

PCS100“import”进一步说明了模块和包的使用方法和注意事项。

“CDROM”这种定量字串怎么融合到类中呢？还有，怎么令交互式命令行软件一运行就会给点提示什么的?

小白的想象力随着不断地尝试成功，也不断地膨胀起来，那么就研究文档吧……

PCS201“cmd”详细描述了 cmd 模块功能，读者可以进一步了解此模块的所有功能。
这是一个辅助用户快速完成一个交互式命令行环境的强力模块，在线相关文档是：
http://aspn.activestate.com/ASPN/docs/ActivePython/2.4/python/lib/module-cmd.html
精巧地址：
http://bit.ly/bVos1

pycdc-v0.5.py

研究和改造的成果如下：

```
# -*- coding: utf-8 -*-
import sys, cmd
from cdctools import *
class PyCDC(cmd.Cmd):
    def __init__(self):
        cmd.Cmd.__init__(self)              # initialize the base class
```

```
 7        self.CDROM = '/media/cdrom0'
 8        self.CDDIR = 'cdc/'
 9        self.prompt="(PyCDC)>"
10        self.intro = '''PyCDC0.5 使用说明:
11    dir 目录名     #指定保存和搜索目录，默认是 "cdc"
12    walk 文件名    #指定光盘信息文件名，使用 "*.cdc"
13    find 关键词    #使用在保存和搜索目录中遍历所有.cdc 文件，输出含有关键词的行
14    ?          # 查询
15    EOF        # 退出系统，也可以使用 Crtl+D(Unix)|Ctrl+Z(Dos/Windows)
16        '''
17    def help_EOF(self):
18        print "退出程序 Quits the program"
19    def do_EOF(self, line):
20        sys.exit()
21
22    def help_walk(self):
23        print "扫描光盘内容 walk cd and export into *.cdc"
24    def do_walk(self, filename):
25        if filename == "":filename = raw_input("输入 cdc 文件名:: ")
26        print "扫描光盘内容保存到:'%s'" % filename
27        cdWalker(self.CDROM,self.CDDIR+filename)
28
29    def help_dir(self):
30        print "指定保存/搜索目录"
31    def do_dir(self, pathname):
32        if pathname == "": pathname = raw_input("输入指定保存/搜索目录: ")
33        self.CDDIR = pathname
34        print "指定保存/搜索目录:'%s' ;默认是:'%s'" % (pathname,self.CDDIR)
35
36    def help_find(self):
37        print "搜索关键词"
38    def do_find(self, keyword):
39        if keyword == "": keyword = raw_input("输入搜索关键字: ")
40        print "搜索关键词:'%s'" % keyword
41
42 if __name__ == '__main__':     # this way the module can be
43    cdc = PyCDC()
44    cdc.cmdloop()
```

运行结果如图 CDay–2-3 所示。

```
~/LovelyPython/CDays/cday-2$ python pycdc-v0.5.py
PyCDC0.5 使用说明:
    dir 目录名      #指定保存和搜索目录，默认是 "cdc"
    walk 文件名     #指定光盘信息文件名，使用 "*.cdc"
    find 关键词     #遍历搜索目录中所有.cdc文件，输出含有关键词的行
    ?           # 查询
    EOF         # 退出系统，也可以使用Crtl+D(Unix)|Ctrl+Z(dos/windows)

(PyCDC)>wakl
*** Unknown syntax: wakl
(PyCDC)>walk try.cdc
扫描光盘内容保存到:'try.cdc'
(PyCDC)>
```

图 CDay–2-3　pycdc-v0.5.py 运行结果

完全与预想吻合!

1. `from cdctools import *` 可以引入已有脚本中的所有函式。
2. 在恰当的位置追加 `cdWalker(self.CDROM，self.CDDIR+filename)` ，就完成了实际功能的绑定。
3. `self`. 在类声明中应该是代表自个儿，代表实体化后的那个自个儿。
4. 类的定量应该都在`__inti__(self)` 初始化时声明。
5. `cmd` 的 `prompt` 就是命令行环境的前缀字串。
6. `cmd` 的 `intro` 果然是命令行环境的最初提示。
7. 括起来的多行字串，可以被自然地记录和打印到命令行环境中。

实现 grep

两大功能之一（扫描光盘信息记录为文件）通过：`dir` 和 `walk` 命令都实现了，现在就差搜索了，其实就是像一个古老的软件 `grep` 一样，打开所有符合要求的文件，读取每一行，如果有指定关键词在行内就打印输出到屏幕……

结合已有的经验，可以非常简单地实现!

```
1 # -*- coding: utf-8 -*-
2
3 def cdcGrep(cdcpath,keyword):
4     filelist = os.listdir(cdcpath)       # 搜索目录中的文件
5     for cdc in filelist:                 # 循环文件列表
6         if ".cdc" in cdc:                # 过滤可能的其它文件,只关注.cdc
7             cdcfile = open(cdcpath+cdc)        # 拼合文件路径，并打开文件
8             for line in cdcfile.readlines():   # 读取文件每一行，并循环
```

```
 9              if keyword in line:          # 判定是否有关键词在行中
10                  print line               # 打印输出
```

对于小白来说，在这个自制的最简单的 grep 功能中，6 行代码中的 5 行都可以自然地写出，只有一行判定代码是新的：

```
if keyword in line:
```

in 也是计算，是 Python 中所有数据结构都支持的一种基础的方便的运算。可以在 Python 交互行命令行环境中快速测试一下：

```
>>> a='123456'
>>> '1' in a
True
>>> a=(1,2,3,4)
>>> 1 in a
True
>>> a=[1,2,3,4]
>>> 1 in a
True
>>> a={1: 11,2: 22,3: 33}
>>> 1 in a
True
```

grep 是古老实用且高效的模式文本匹配工具，在所有的 Unix/Linux 系统中都会默认安装，它最常做的事儿是将一堆文本中包含某个模式的文本行找出来，如：

```
~$ cat /proc/cpuinfo | grep core
core id         : 0
cpu cores       : 2
core id         : 1
cpu cores       : 2
```

小结

通过对 cmd 模块的边学边用，小白今天至少可以体验到：

1. 类
2. 多行注释
3. 模块
4. 模块自测
5. 字串包含计算

虽然获得的可用代码的行数一下子冲破了个位数，但是 90% 的代码都可以在头两天的 4 行代码中找到模式。

练习

1. 在前文的 grep 实现例子中，没有考虑子目录的处理方式，因为如果直接 open 目录进行读

操作，会出现错误，所以请读者修改这个示例代码，以便考虑到子目录这种特殊情况，然后把最后探索出的 `cdcGrep()`嵌入 pycdc-v0.5.py 中，实现完成版本的 PyCDC。

提示：子目录处理，可以先判断，如果是子目录，就可以递归调用 `cdcGrep()`函式。

2. 编写一个类，实现简单的栈。数据的操作按照先进后出（FILO）的顺序。主要成员函式为 `put(item)`，实现数据 item 插入栈中；`get()`，实现从栈中取一个数据。

CDay–1　实用化中文

中文处理，完成功能的实用化

你碰到 99%的问题，其他人之前已经遇到过了，所以，最佳的解决方式就是找到那段别人解决相似问题的代码!

回顾需求

小白已经实现的需求可以实现如下功能。

1. 可以扫描光盘内容，并存储为硬盘上的文本文件。
 1）存储成*.cdc 的文本文件；
 2）快速指定保存目录；
 3）快速指定保存的文件名。
2. 可以根据储存到硬盘上的光盘信息进行搜索。
 1）搜索指定目录中所有*.cdc 文件；
 2）指定关键字进行搜索；
 3）列出所有含有关键字的信息行。

进一步尝试

回想起来，一直尝试搜索的都是英文关键字，中文的没有试过。

尝试来几下！……呜乎，什么也查不出来!

查阅记录文本

先来看看图 CDay–1-1。

mymusic-1.cdc

```
1
2 /media/cdrom0;['Akira', 'BERSERK', 'FLCL', 'GHOST IN THE SHELL',
'JIN-ROH', 'LAIN', '????', '????PATLABOR', '????', '????'];[]
3 /media/cdrom0/Akira;[];['Track01.mp3', 'Track02.mp3',
'Track03.mp3', 'Track04.mp3', 'Track05.mp3', 'Track06.mp3',
'Track07.mp3', 'Track08.mp3', 'Track09.mp3', 'Track10.mp3']
4 /media/cdrom0/BERSERK;['?????????.files'];['barsark_dagame_02.mp3',
'barsark_dagame_04.mp3', 'barsark_dagame_05.mp3',
'barsark_dagame_06.mp3', 'barsark_dagame_07.mp3',
'barsark_dagame_08.mp3', 'barsark_dagame_09.mp3',
'barsark_dagame_10.mp3', 'barsark_dagame_11.mp3',
```

图 CDay–1-1　mymusic-1.cdc 内容

这种数据对吗？

当初为了简单，使用文档中的基本型，即：

```
#'cdctools.py' 中 cdWalker(cdrom,cdcfile) 的动作
...
   for root, dirs, files in os.walk(cdrom):
      export+="\n %s;%s;%s" % (root,dirs,files)
...
```

就是使用 `os.walk()` 的天然输出组织成每一行：

```
/media/cdrom0/EVA/Death-Rebirth;[];['eva8-01.Mp3', 'eva8-02.Mp3',...]
    ^                            ^ ^ ^
    |                           | | +- files 列表，此目录的文件名
    |                           | +- 各个数据段使用";" 分隔
    |                           +- dirs 列表，子目录名，如果没有就为空
    +- 当前目录
```

瞧着格式顶像，为什么到中文的地方就成了问号呢？

中文！永远的痛

不问不知道，一把辛酸泪啊！在网络中一搜索才知道，只要是个中国人，不论整什么开发，中文!永远会遇到各种问题的。不过，幸好比小白勤劳的人海了去，有关中文的 Python 处理建议一搜一大堆。但是，**有时候，选择太多也是个问题**。

编码问题

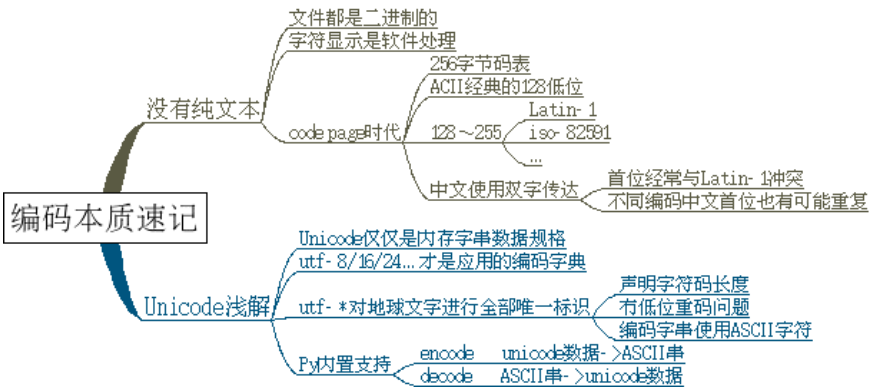


图 CDay–1-2　编码思维图谱

有行者给出如图 CDay–1-2 所示的思维图谱（Mind Map），目前还在理解过程中，先使用已知的方式测试本地硬盘文件的目录情况，如图 CDay–1-3 所示。

```
~/LovelyPython/CDays/cday-1$ ipython
Python 2.5.2 (r252:60911, May  7 2008, 15:19:09)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.4 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

In [1]: import os

In [2]: for root, dirs, files in os.walk("./中文编码"):
   ...:     print "\n%s;%s;%s " % (root, dirs, files)
   ...:

./中文编码;[];['\xe4\xb8\xad\xe6\x96\x87\xe7\xbc\x96\xe7\xa0\x81\xe6\xb5\x8b\xe8
\xaf\x95', '\xe4\xbd\xa0\xe5\xa5\xbd']
```

图 CDay–1-3　本地文件目录的测试结果

看着就不同，还得根据理解继续尝试，从而知道自己是否真正正确理解，另外一些测试结果如图 CDay–1-4 所示。

```
In [3]: unicode([dirs for dirs in os.walk("./中文编码")][0][0], 'utf8')
Out[3]: u'./\u4e2d\u6587\u7f16\u7801'

In [4]: unicode([dirs for dirs in os.walk("./中文编码")][0][0], 'utf8').encode("
utf8")
Out[4]: './\xe4\xb8\xad\xe6\x96\x87\xe7\xbc\x96\xe7\xa0\x81'

In [5]: print unicode([dirs for dirs in os.walk("./中文编码")][0][0], 'utf8').en
code("utf8")
./中文编码
```

图 CDay–1-4　另一些测试结果（使用 unicode）

```
unicode(原始文本,  'utf8'  ).encode('utf8')
文本 ==decode()--> [unicode] ==>encode()--> utf-8 文本
 ^      ^             ^        ^             ^
 |      |             |        |             +- 最终的渴求
 |      |             |        +- 此为编码过程;可以从 unicode 输出为任意编码
 |      |             +- Python 内置支持的 unicode 格式数据
 |      +- 此为解码过程,将已知编码的文本编译成宇宙通用的 unicode 数据
 +- 原始文本信息,是什么编码你得知道!
```

也就是说文件没有编码之说，其实都是按二进制格式保存在硬盘中的，仅仅是在写入读取时须使用对应的编码进行处理，以便操作系统配合相关软件/字体，绘制到屏幕中给人看。所以关键问题是得知道原先这些字串数据是使用什么编码来编译的！但是在 Unicode 之前都是使用类似对照表的形式来组织编码的，无法从串数据流本身统一解出不同的文字来。

只有猜！

猜编码函式一

```
def _smartcode(stream):
    try:
        ustring = unicode(stream, 'gbk')
    except UnicodeDecodeError:
        try:
            ustring = unicode(stream, 'big5')
        except UnicodeDecodeError:
            try:
                ustring = unicode(stream, 'shift_jis')
            except UnicodeDecodeError:
                try:
                    ustring = unicode(stream, 'ascii')
                except:
                    return u"bad unicode encode!"
```

收集的音乐可不仅仅中文的，印度、伊朗等国的音乐都有可能收集，如果逐一去尝试，这个判定树就太可怕了！

升级！使用 Python 的数据对象进行集成！

猜编码函式二

```
def _smartcode(stream):
    tryuni = ("gbk"
          ,"gb2312"
```

```
            ,"gb18030"
            ,"big5"
            ,"shift_jis"
            ,"iso2022_kr"
            ,"iso2022_jp"
            ,"ascii")
      try:
         for type in tryuni:
            try:
               ustring = unicode("%s}"%type+stream, type)
               #try decode by list
            except:
               continue
               #break!continue try decode guess
      except:
         return u"bad unicode encode!"
```

但是……

PCS6 PCS6“Python 与中文”进一步全面地阐述了在 Python 中面对中文数据时的思路和技巧。

chardet

这么一项项猜，还是显得很傻，万一有些字的高位在不同编码中是相同的，那真的是只能撞大运了！尤其是对可怜的难兄难弟：gbk 和 big5。

PCS112 PCS112“异常”讲述了 Python 中异常的使用，即 try ... except ...的使用等。

比如：

```
>>> print '变巨'.decode('big5')
```

曹操

```
>>> print '变巨'.decode('gbk')
```

变巨

再问行者，他们给出个地址：*http://chardet.feedparser.org/*，上面有 Character encoding auto-detection （自动字符探测器）。

真的有呀！而且是 Mozilla 使用的！立用不疑！

怎么安装外部模块呢？软件包下载，解开压缩，嗯？没有 INSTALL 说明文件，但是有个 setup.py ，尝试执行一下（如图 CDay–1-5 所示）。

```
~/ubuntu_software/python/chardet-1.0.1$ ls -l
总用量 44
drwxr-xr-x 3 shengyan shengyan  4096 2008-04-11 19:03 build
drwxr-xr-x 2 shengyan shengyan  4096 2008-04-11 19:05 chardet
-rwxr-xr-x 1 shengyan shengyan 26432 2008-03-05 13:46 COPYING
drwxr-xr-x 4 shengyan shengyan  4096 2008-03-05 13:46 docs
-rwxr-xr-x 1 shengyan shengyan  1983 2008-03-05 13:46 setup.py
~/ubuntu_software/python/chardet-1.0.1$ python setup.py
usage: setup.py [global_opts] cmd1 [cmd1_opts] [cmd2 [cmd2_opts] ...]
   or: setup.py --help [cmd1 cmd2 ...]
   or: setup.py --help-commands
   or: setup.py cmd --help

error: no commands supplied
~/ubuntu_software/python/chardet-1.0.1$ sudo python setup.py install
running install
running build
running build_py
running install_lib
running install_egg_info
Removing /usr/lib/python2.5/site-packages/chardet-1.0.1.egg-info
Writing /usr/lib/python2.5/site-packages/chardet-1.0.1.egg-info
```

图 CDay–1-5　chardet 安装示意

```
-$1s-l
总用量为 33。
drwxr-xr-x 3 zoomq zoomq    72 2008-04-29 11：25 build
drwx------ 2 zoomq zoomq  1264 2006-01-11 01：34 chardet
-rwx------ 1 zoomq zoomq 26432 2006-01-11 01：34 COPYING
drwxrwxrwx 4 zoomq zoomq   296 2006-01-11 01：34 docs
-rwx------ 1 zoomq zoomq  1981 2006-01-11 01：34 setup.py
~$ python setup.py
usage：  setup.py [global_opts] cmd1 [cmd1_opts] [cmd2 [cmd2_opts] ...]
   or：  setup.py --help [cmd1 cmd2 ...]
   or：  setup.py --help-commands
   or：  setup.py cmd --help

error：  no commands supplied
~$ sudo python setup.py install
running install
running build
running build_py
running install_lib
running install_egg_info
Removing /usr/lib/python2.5/site-packages/chardet-1.0.egg-info
Writing /usr/lib/python2.5/site-packages/chardet-1.0.egg-info
```

so easy！好像使用了类似小白已经完成的交互式提醒呀！

看来所有 Python 的软件都可以通过 `python setup.py install` 进行安装！

PCS202 “ chardet ” 进一步说明了此外部模块的信息，并分享了使用体验。

```
 1 import chardet
 2 def _smartcode(stream):
 3     """smart recove stream into UTF-8
 4     """
 5     ustring = stream
 6     codedetect = chardet.detect(ustring)["encoding"]
 7     print codedetect
 8     try:
 9         print ustring
10         ustring = unicode(ustring, codedetect)
11         print ustring
12         return "%s %s"%("",ustring.encode('utf8'))
13     except:
14         return u"bad unicode encode try!"
```

经过测试，已确信它在各种情况下都可以正确识别！但是不论怎么尝试，已经保存下来的.cdc 文本依然是 `ASCII` 码！

另外的思路

也许真的就是 ASCII 码呢？

幸福的自由

冷静一下，既然，文件没有编码，都是二进制的，那么光盘文件是如何被系统认识的？！在列表中吼了一下，行者有点无奈地说：“TiosnG！”——`There is one site named Google!`
好吧，小白老实地搜索了一番，发现了 `iso9660` —所有光盘基本都是此文件格式的，同 M$的 FAT32/ntfsGNU/Liunx 使用的 ext2/3、Unix 使用的 UFS...一样只是种文件系统。那么它必然是通过某种软件进行转换从而可以访问的。在 Ubuntu 中当然是标准地调用朴实强大的 `mount` 命令了。

```
~$ cat /etc/mtab
...
/dev/sda3 /dos vfat rw,utf8,umask=007,gid=46 0 0
...
/dev/sda1 /windows ntfs rw,nls=utf8,umask=007,gid=46 0 0
/dev/scd0 /media/cdrom0 iso9660 ro,noexec,nosuid,nodev,user=zoomq 0 0
...
```

观察默认的挂接光盘的行为和挂接 Dos 和 Windows 分区的情况，感觉是默认的挂接行为没有使用编码处理。那么通过改变挂接的行为，应该可以改善读取不到正确文本的情况。

```
#挂接成 GBK
~$ sudo mount -o ro,norock,iocharset=cp936 /dev/scd0 /media/cdrom0
#挂接成 UTF8
~$ sudo mount -o ro,norock,iocharset=utf8 /dev/scd0 /media/cdrom0
```

果然！果然！相对 M$，系统安装成什么语言的系统，就永远使用该系统强行挂接光盘，好像八成应该不可以轻易按照你的意愿改变的……以前小白对于自由软件的认识仅仅是免费，难用，这下忽然间有所顿悟，不觉写下感慨：

自由软件好，我用我自在！

所谓自由，是基于对自个儿的了解，真正理解了自个儿想要什么之后，自由软件支持你的一切尝试!哈哈哈!

小白可以感受到这些，应该算是**小灰**了，可以稳定地向成熟的**小黑**——一名黑客挺进了！

提示：Hacker（黑客）绝对不是中国媒体中宣传的那些攻击他人电脑的家伙，黑客是些创造技术奇迹的单纯的人们（*http://wiki.woodpecker.org.cn/moin/HackerHowto*，精巧地址：*http://bit.ly/31rUuY* ）。被翻译所误指的那类家伙是 Cracker（骇客）（*http://en.wikipedia.org/wiki/Cracker*，精巧地址：http://bit.ly/3Xx0A），他是破坏者，未经授权而企图进入电脑系统者。这种入侵者通常会恶意进入他人的系统，而且有许多技巧可以破坏他人的系统。这个名词是骇客（Cracker）在 1985 年为对抗新闻媒体滥用 hacker 而提出的。1981~1982 年前，曾有人推动使用“毛虫”代表 Cracker，但并未成功。

改善数据结构

查询是可以了，但是，使用默认的列表打印格式来存储和汇报实在不咋的，想修改修改，于是

```
#cdctools.py...
def formatCDinfo(root,dirs,files):
    export = "\n"+root+"\n"
    for d in dirs:
        export+= "-d "+root+_smartcode(d)+"\n"
    for f in files:
        export+= "-f %s %s \n" % (root,_smartcode(f))
    export+= "="*70
```

```
    return export
```

存储下来的 .cdc 片段为：

```
...
-f /media/cdrom0/RyokoHirosue/Ryoko Hirosue-files.files  title.gif
========================================================================
/media/cdrom0/RyokoHirosue/成长物语
-d /media/cdrom0/RyokoHirosue/成长物语 音乐极限--成长物语.files
-f /media/cdrom0/RyokoHirosue/成长物语  RH991101.mp3
-f /media/cdrom0/RyokoHirosue/成长物语  RH991102.mp3
```

d 代表目录；f 代表文件。它们分别是 directory（目录）和 file（文件）的缩写。

PCS107 “字串格式化”进一步介绍了用来快速重构光盘信息文本格式的 Python 所内置的字符模板支持特性。

小结

这日，小白仅仅解决了一个问题——中文搜索问题；解决的尝试路径及相互关系如图 CDay–1-6 所示。但是，实际上他获得了两大方面的进步：

1. 有了文本编码方面的知识。
2. 问题解决的方法论境界提高了一个层次，开始使用系统级别的解决方案了。

相对 Python 方面，仅仅追加了一对内置函式和一个外部模块包使用的体验。

关键词：

编码

unicode

chardet

mount

注意：如果你使用 M$ 系统，运气好的话是不会见到笔者列出的现象的，但是不保证以后遇不到自动处理下搞不定的光盘内容。希望那时候小白真的有个自由环境可以尝试其他方案。

提示：事实上存在 Windows 下面的完全 Unix 环境，如 Cygwin（*http://www.cygwin.com*），它是一个通过运行于 Windows 下的免费的 Unix 子系统使用一个 Dll（动态链接库）来实现的虚拟机，可以直接在 Windows 环境中使用各种 Unix 实用工具。

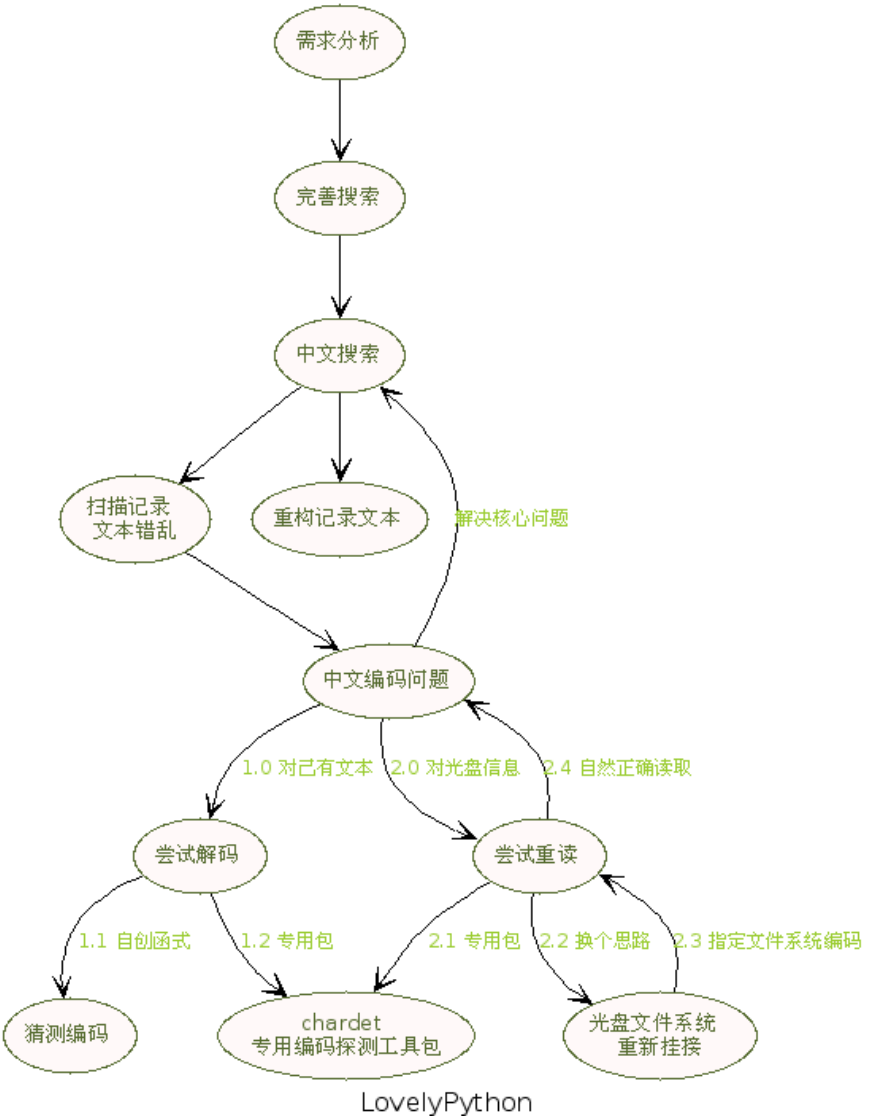


图 CDay–1-6　CDay–1 问题解决思路图谱

练习

1. 自动判定你自个儿或是朋友的 Blog 是什么编码的？
2. 如果是非 utf-8 的，请编写小程序自动将指定文章转换成 utf-8 编码保存。

CDay0　时刻准备着！发布

感受软件工程进行发布的准备

发布！为了全人类！因为一个人如果力求完善自己，他就会明白同时也必须完善他人。一个人如果不关心别人的完善，自己便不可能完善。

从需求导出用户群

小白已经实现的功能如下所示。

1. 扫描光盘内容并存储为硬盘上的文本文件。
 1） 存储成*.cdc 的文本文件；
 2） 可以快速指定保存目录；
 3） 可以快速指定保存的文件名。
2. 根据储存到硬盘上的光盘信息计件进行搜索。
 1） 可以搜索指定目录中所有*.cdc 文件；
 2） 可以指定关键字进行搜索。
 - 列出所有含有关键字的信息行。

对于大胆并逐渐习惯命令行界面的小白来说，现在的 PyCDC 基本可用了。那么这样的小工具，有哪些人会愿意使用？推想一下：

- 有很多光盘的人
- 知道并会安装 Python 的人
- 不惧怕命令行的人

- 勇于尝试的人

这样的人也必定是愿意分享的人，小白受行者的熏陶，也想把刚刚完成的可用的 PyCDC 汇报给列表中的前辈们使用，一是想获得夸奖，不过更加希望有人通过使用反馈意见，甚至于代码的改进意见，发布出来，进而可以将自己首个作品长期发展下去，造福人类。

所以要发布！

发布的本质

发布好像不是通告一嗓子就可以了的事儿。

给人用

小白尝试将 PyCDC 整理了个压缩包发送给朋友，结果所有的朋友都说不会使用……郁闷是小白的真实写照。

只好再次到列表中询问：“怎么发布软件啊？！”行者回复：“文档！文档！文档！”

文档

分，分，分！学生的命根！ 文档，文档，文档！软件的颜面！

软件是给人使用的，但是你无法跑到每个用户身边，去演示如何使用你的软件吧?！所以，文档！友好的文档！清晰的文档！有效的文档！ 就是作者的代言人、形象大使，是潜在参与者/体验者的向导，是帮助他人学习/使用/改进你的软件的实用拐杖！对于使用相关许可证进行源代码发布的自由软件来说，文档也是工程的一部分！面对乱七八糟的文档，是没有人敢于使用你的软件的。

再次开发

所以，发布意味着针对你期望的用户再次开发！

标准的自由软件包应该有哪些文档?

1. `AUTHORS` 作者自述
2. `LICENSE` 许可证类型（Python 世界喜欢简单的 BSD 系列许可证）

3. `README` 软件说明
4. `ChangeLog` 修订历史
5. `PKG-INFO` 包信息，提供给一些自动程序使用

这些都是必需的类似八股文的内容。更加重要的是，小白对自个儿经过反复尝试总结出来的多个小巧核心功能，非常期望他人可以重复使用以下函式！

- `cdWalker()`
- `cdcGrep()`
- `_smartcode()`

那么如何更友好地说明如何单独使用这些函式呢？进而，随着代码的频繁变化，怎样才能令这些函式便于维护？

问行者，才知道原来这叫**文档化开发**——将软件 API 文档同代码结合起来写，写代码的同时也完成了文档，它是种常用的开发、维护同时进行的开发模式。

- **文档化开发**的主要技巧，就是利用注释！
- 但是使用什么格式的注释？注释是通过什么工具转换成文档的？

行者又推荐了 epydoc。

PCS104

PCS104“ 注释 ”进一步介绍了 Python 中注释的写法，配合注释可以自动进行开发文档生成的工具……

epydoc

epydoc，这是一个轻松的 Py 文档生成器（Easy Py Documentor）！其网址为：*http://wiki.woodpecker.org.cn/moin/CodeCommentingRule*（精巧地址：*http://bit.ly/46BkrB*）

在网址 *http://epydoc.sourceforge.net/api/epydoc-module.html*（精巧地址：*http://bit.ly/250krV*）上可以看到，epydoc 自个儿的文档输出，是多么专业和规范！

在简要学习相关知识后，小白发觉只要起草一个简单的配置文件就可以了：

```
##可以命名为 epydoc.cfg
[epydoc]
# 项目信息
name: PyCDC
url: http://wiki.woodpecker.org.cn/moin/ObpLovelyPython

# 想进行处理的文件或是模块，一般指定目录即可
modules: /pat/to/pycdc.py，/pat/to/cdctools.py

# 使用 html 格式，输出文档到 apidoc/ 目录中
output: html
```

```
    target: apidocs/

    # 指定 Graphviz dot.所在的位置，可以生成各种 UML 关系图谱（又一项自由环境中的福利）！
    graph: all
    dotpath: /usr/bin/dot
```

这样一来，小白就可以随时使用类似于“~$ epydoc --config epydoc.cfg”的命令获得最新的 API 文档，效果如图 CDay0-1 所示：

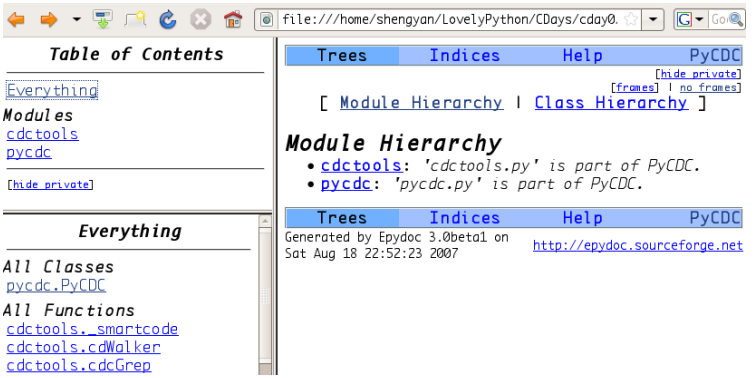


图 CDay0-1　epydoc 生成的 API 文档

PCS203 “ epydoc ”进一步介绍了这一优秀的文档化开发支持工具；所谓文档化开发，就是将软件工程过程中的相关文档，嵌入到相应代码的注释中，通过简要的字串约定，然后由工具自动摄取出相应的文本，组织成可以随时查阅的文档。

它是非常类似 Java DOC 带框架的综合视图的文档站点！

引发的改进

函式命名的合理性

小白自个儿之前随手命名的函式，在注释中，前后看起来并不顺畅，回想一路使用着的 Python 内置函式命名：

1. os.listdir()
2. os.walk()
3. encode()
4. open().write()

它们是多么地直接啊，小白现在才发现，命名也是大学问，于是查看了 Python 开发编码规范（*http://wiki.woodpecker.org.cn/moin/PythonCodingRule*）。

简单地说就是要应用前人在大量的代码编写中摸索出来的规范，这些规范可以令任何人

快速在代码中区分出各种关键变量 / 类 / 函式等。

有关命名原则，小白现在可以理解并习惯了。

1. 各种命令尽量使用动宾式短语。
2. 目录 / 文件名全部小写。
3. 类名使用首字母大写单词串（`WikiNames`）。
4. 全局变量使用全大写字串（`CAPWORD`）。
5. 函式使用混合大小写串 （`mixedCase`）。
6. 内部变量、常数，全部小写。

综合应用就形如以下代码。

```
1 from mypacket import MyClass
2 GLOBALVAR="是也乎"
3 def doInsert(arg):
4    MyClass.myFunc(GLOBALVAR)
```

所以，小小修订一些变量 / 函式的命名，使之更加贴切是作者在发布前的礼貌。

```
cdctools._smartcode(stream)-> cdctools._smartcode(ustring)
codedetect = chardet.detect(ustring)["encoding"]->codename
```

PCS7

PCS7“Python 编码规范”完整地翻译了 PEP-0008 Guido 原创的"Python 风格指南" http://www.python.org/dev/peps/pep-0008/（精巧地址:http://bit.ly/2OQ8Z3) 这是写出靠谱的 Python 脚本的基础素养,至少应该认真反复精读 3 回。

小结

通过本日故事小白正式登堂入室到软件工程世界，**沟通需要代价，改进由我做起！**

- 既然软件的发布就是和潜在的参与者沟通，沟通就需要良好的媒介，在软件世界，软件工程文档是最好的中介；
- 自由软件世界，聪明人太多，连文档也有专门的辅助工具，小白快乐地使用并体验着；
- 对 epydoc 的体验将引导小白思考很多实现功能以外的事儿，这让小白越来越强了，行动更加有章法，他正向靠谱的程序员健康挺进……

练习

请根据软件发布的流程和软件开发的编码规范，将读者在前面章节所写的程序修改并发布出去。另外，可以查找下除了 epydoc 外还有哪些较好的 py 文档生成器？

CDay+1　优化！对自个儿的反省

首次重构优化数据结构

没有最好，只有更合理！

扩展需求

到目前为止，小白已经实现的功能为：

1. 扫描光盘内容并存储为硬盘上的文本文件。
 1） 存储成*.cdc 的文本文件 `cdctools.cdWalker()`；
 2） 可以快速指定保存目录 `PyCDC.do_dir()`；
 3） 可以快速指定保存的文件名 `PyCDC.do_walk()`。
2. 根据储存到硬盘上的光盘信息文件进行搜索。
 1） 可以搜索指定目录中所有*.cdc 文件 `cdctools.cdcGrep()`；
 2） 可以指定关键字进行搜索 `PyCDC.do_find()`。
 - 列出所有含有关键字的信息行。

而且，小白已经组织好了文档化的注释，足以让有心人快速了解程序整体的结构，复用相关函式，或是掺和持续改进。这不，还真有另外的小白提出了问题。

问一声

“为什么叫 PyCDC？和 CD Catalog Expert 有关系吗？”有好事者先置疑了名称。

小白心想：先随便起的名字，含义应该是 “CD Collector”或是“CD Commander”，难道已经有同类甚至同名的软件了？搜索一下，找到这个网址：*http://www.zero2000.*

com/cd-catalog-expert/index.html（精巧地址：*http://bit.ly/17N5fU*）。

咦，还真有 CD Collection，网址为 *http://www.nicomsoft.com/cdc/*（精巧地址：*http://bit.ly/4vsuwe*），甚至于，类似通过抓取指定介质的信息进行外部管理的软件是一个大类，叫 Disk Catalog。

Disk Catalog

有关磁盘的分类，找到的评比文章为：*http://tech.sina.com.cn/c/2003-12-05/25149.html*（精巧地址：*http://bit.ly/3U8vbr*）。

而小白拍脑袋想出来的文本格式的光盘信息摘要形式，事实上是历史悠久的。

- 完全类似 Total Commander 的 DiskDir、CatalogMaker 等插件生成的 Catalog 摘要。
- 而在 Linux 下面，FTP 服务器自古就提供通过 `ls -lR` 生成的文本摘要。

对比及推导

那么哪种比较好？

小白当然愿意令自个儿的小工具发扬光大下去，那么改进是不可避免的！其实记录的摘要文本已经升级过一次了，从 `os.walk()` 输出的自然结构格式为：

```
/media/cdrom0/EVA/Death-Rebirth;[];['eva8-01.Mp3', 'eva8-02.Mp3',...]
    ^                            ^  ^  ^
    |                           |  |  +- files 列表，此目录的文件名
    |                           |  +- 各个数据段使用";" 分隔
    |                           +- dirs 列表，子目录名，如果没有就为空
    +- 当前目录
```

使用专门的 `formatCDinfo()` 函式格式化，为：

```
...
-f /media/cdrom0/RyokoHirosue/Ryoko Hirosue-files.files  title.gif
========================================================================
/media/cdrom0/RyokoHirosue/成长物语
-d /media/cdrom0/RyokoHirosue/成长物语 音乐极限--成长物语.files
-f /media/cdrom0/RyokoHirosue/成长物语  RH991101.mp3
-f /media/cdrom0/RyokoHirosue/成长物语  RH991102.mp3
```

其中 `d` 代表目录；`f` 代表文件。

虽然是自个儿随意设计的格式，但是小白可以体会出以下好处：

1. 格式自然，可以直接人工查阅；
2. 每行包含绝对必要的信息：对象类型（目录？文件？）、 所在目录、文件名；
3. 搜索匹配后的输出行就是自然行，并包含必要信息。

那么也快速考查一下其他数据格式：

Linux 目录摘要命令：`ls -lR`

```
~/LovelyPython/CDays/cday0$ ls -lR
.:
drwxr-xr-x 3 zoomq zoomq  4096 2007-08-18 22:43 apidocs
-rwxr-xr-x 1 zoomq zoomq    87 2007-07-31 14:58 AUTHORS
drwxr-xr-x 3 zoomq zoomq  4096 2007-07-31 14:58 cdc
-rwxr-xr-x 1 zoomq zoomq  3874 2007-08-18 22:42 cdctools.py
-rw-r--r-- 1 zoomq zoomq  3758 2007-08-18 22:42 cdctools.pyc
-rwxr-xr-x 1 zoomq zoomq  1152 2007-08-18 22:02 ChangeLog
-rw-r--r-- 1 zoomq zoomq   694 2007-08-18 22:30 epydoc.cfg
-rwxr-xr-x 1 zoomq zoomq 35148 2007-08-18 22:00 LICENSE
-rwxr-xr-x 1 zoomq zoomq   246 2007-08-18 22:01 PKG-INFO
-rwxr-xr-x 1 zoomq zoomq  3273 2007-08-18 22:28 pycdc.py
-rw-r--r-- 1 zoomq zoomq  4634 2007-08-18 22:28 pycdc.pyc
-rwxr-xr-x 1 zoomq zoomq  1082 2007-07-31 14:58 README

./apidocs:
-rw-r--r-- 1 zoomq zoomq   760 2007-08-18 22:52 api-objects.txt
-rw-r--r-- 1 zoomq zoomq 15247 2007-08-18 22:52 cdctools-module.html
-rw-r--r-- 1 zoomq zoomq 26380 2007-08-18 22:52 cdctools-pysrc.html
-rw-r--r-- 1 zoomq zoomq  4139 2007-08-18 22:52 class-tree.html
-rw-r--r-- 1 zoomq zoomq   340 2007-08-18 22:52 crarr.png
...
```

这个命令是以空行来区分不同目录下的信息的。在这个命令中，每节是固定顺序的信息：

```
./apidocs:  <-- 当前目录
-rw-r--r-- 1 zoomq zoomq   760 2007-08-18 22:52 api-objects.txt
^   ^            ^   ^       ^    ^                   ^
|   |            |   |       |    |                   +- 文件/目录名
|   |            |   |       |    +- 创建/修订时间
|   |            |   |       +- 体积
|   |            |   +- 用户组
|   |            +- 用户
|   +- 权限组
+- 文件(-)或是目录(d)
```

它包含了丰富且规范的信息，但是作为 PyCDC，其中有关时间/权限的信息都是不必要的，而且不是跨平台兼容的，在 M$平台中，权限信息就根本用不上。

CD Catalog Expert 的文本数据格式如下所示：

```
[Info]
DriveType=5
ImagePath=L:
Volume=MCollec.39
Serial=723840260
FAT=CDFS
DiskSize=676020224
DiskFree=0

[Comment]
0= South Park(南方公园)
1= 可儿乐队精选 The Corrs Greathits
2= 林忆莲
3= 痴澄庆《哈林天堂》Harlem'm
4=Heaven

[L:]

[L:\South Park(南方公园)]
South Park - Merry Fucking Xmas.mp3=1986560
South Park - Uncle Fucker.mp3=1056768

...
```

这样一来，小白也看明白了，完全是标准的.ini 配置文本格式!

[Info] 一节是软件和光盘的整体信息；[Comment] 一节是根目录列表；[目录] 各节是各个目录的文件信息。

和自创的文本格式相比这个格式更加规范，而且可以直接由成熟的已有软件解读！小白突然开窍……这等于是直接利用既有客户群！有兼容性的话，原先使用 CD Catalog Expert 的人不也更加有可能使用 PyCDC 了？至于其他使用数据库或是自创二进制文件的软件，小白没有精力和兴趣研究了，就这么着!

快速重构

让 PyCDC 的光盘摘要文本格式，直接向已有软件的数据格式看齐!

问题探索

有了目标就有了研究的动力！小白立即根据分析结果确定了代码难点：

1. 怎么处理 .ini 文件？
2. 怎么获得文件的大小？

老规矩，在列表中吼!

结果，行者们给出来的答案是一致的：内置模块有支持！

呜乎……看来犯了依赖病，小白重新拿起《提问的智慧》BS 一下自个儿，然后，翻开 Global Module Index（全局模块索引）这一大通乱找，果然看到了！

ConfigParser

`ConfigParser`（配置处理机）?! 这也忒直白了吧!

PCS204“ ConfigParser ”进一步详细地分享了使用这一常用内建模块的技巧。

粗略地看了一下，也就 RawConfigParser-objects（ 原始配置处理机对象）有写出到文件的函式，就它了！在 iPython 中快速尝试使用一下（如图 CDay1-1 所示）。

```
In [8]: from ConfigParser import RawConfigParser as rcp

In [9]: cfg = rcp()

In [10]: cfg.add_section("Info")

In [11]: cfg.set("Info", "ImagePath", "path/to/my/cdrom")

In [12]: cfg.set("Info", "foo", "CD 信息")

In [13]: cfg.write(open("try.ini", "w"))

In [14]: cat try
try      try.ini

In [14]: cat try.ini
[Info]
imagepath = path/to/my/cdrom
foo = CD 信息
```

图 CDay1-1　ConfigParser 使用示意

果然好使!

PCS2“ 交互环境之 iPython ”分享了这一优秀的可以提高工作效率的环境体验!

注意：图 CDay1-1 截取的是真实的尝试过程，不过由于笔者使用的是增强的 Python 交互环境“iPython”，所以，在图的[14]处，可能会让读者有所误解，这里是使用 tab （制表符键）时，iPython 自动给出的猜测输出，以便用户选定/回忆出真正想要的对象/文件等，这里并不像其他交互环境中使用 Enter （回车键）！

文件?是 os 的

文件大小信息在哪？以 f 开头的模块都不像呀！小白突然想起来，这文件和目录都是文件系统，应该是操作系统管理的。比如光盘扫描的 `walk()`函式就是在 os 模块中的，文件这个信息也应该在那里?

Files and Directories（文件及目录）果然！有个 `os.stat()` 系统状态函式!

照例在 iPython 中快速尝试使用一下（如图 CDay1-2 所示）。

```
In [17]: import os

In [18]: os.stat("./ChangeLog")
Out[18]: (33261, 1180964L, 2052L, 1, 1000, 1000, 1460L, 1221903012, 1209130528,
1221877926)

In [19]: os.stat("./ChangeLog").st_size
Out[19]: 1460L

In [20]: ls -lF
总用量 92
drwxr-xr-x 3 shengyan shengyan  4096 2008-04-25 21:35 apidocs/
-rwxr-xr-x 1 shengyan shengyan    87 2008-04-25 21:35 AUTHORS*
drwxr-xr-x 3 shengyan shengyan  4096 2008-04-25 21:35 cdc/
-rwxr-xr-x 1 shengyan shengyan  4260 2008-04-25 21:35 cdctools.py*
-rw-r--r-- 1 shengyan shengyan 10986 2008-04-25 21:35 cdctools-utf8-beautify.cdc
-rwxr-xr-x 1 shengyan shengyan  1460 2008-04-25 21:35 ChangeLog*
-rw-r--r-- 1 shengyan shengyan   694 2008-04-25 21:35 epydoc.cfg
-rwxr-xr-x 1 shengyan shengyan 35148 2008-04-25 21:35 LICENSE*
-rwxr-xr-x 1 shengyan shengyan   246 2008-04-25 21:35 PKG-INFO*
-rwxr-xr-x 1 shengyan shengyan  2930 2008-04-25 21:35 pycdc.py*
-rwxr-xr-x 1 shengyan shengyan  1082 2008-04-25 21:35 README*
-rw-r--r-- 1 shengyan shengyan    53 2008-09-20 17:31 try.ini
```

图 CDay1-2　os.stat()使用示意

Bingo! 就它了!

PCS200

PCS200 “ os(.stat;.path) ” 中名为 os 的内建模块里不仅仅有 stat()，还有其他大量的有关操作系统的好用工具!

完成重构

简单地说就是利用基础配置处理机模块，将原先 `os.walk()` 生成的信息组织成类 ini 的文本保存下来：

```
1 def cdWalker(cdrom,cdcfile):
2     '''光盘扫描主函式
3     @param cdrom: 光盘访问路径
4     @param cdcfile: 输出的光盘信息记录文件(包含路径,绝对、相对都可以)
5     @return: 无,直接输出成*.cdc 文件
6     @attention: 从 v0.7 开始不使用此扫描函式,使用 iniCDinfo()
7     '''
8     export = ""
```

```
    for root, dirs, files in os.walk(cdrom):
        print formatCDinfo(root,dirs,files)
        export+= formatCDinfo(root,dirs,files)
    open(cdcfile, 'w').write(export)

def formatCDinfo(root,dirs,files):
    '''光盘信息记录格式化函式
    @note: 直接利用 os.walk() 函式的输出信息进行重组
    @param root: 当前根
    @param dirs: 当前根中的所有目录
    @param files: 当前根中的所有文件
    @return: 字串,组织好的当前目录信息
    '''
    export = "\n"+root+"\n"
    for d in dirs:
        export+= "-d "+root+_smartcode(d)+"\n"
    for f in files:
        export+= "-f %s %s \n" % (root,_smartcode(f))
    export+= "="*70
    return export
```

```
def iniCDinfo(cdrom,cdcfile):
    '''光盘信息.ini 格式化函式
    @note: 直接利用 os.walk() 函式的输出信息由 ConfigParser.RawConfigParser
           进行重组处理成 .ini 格式文本输出并记录
    @param cdrom: 光盘访问路径
    @param cdcfile: 输出的光盘信息记录文件(包含路径,绝对、相对都可以)
    @return: 无,直接输出成组织好的类.ini 的*.cdc 文件
    '''
    walker = {}
    for root, dirs, files in os.walk(cdrom):
        walker[root]=(dirs,files)    # 这里是个需要理解的地方
    cfg = rcp()
    cfg.add_section("Info")
    cfg.add_section("Comment")
    cfg.set("Info", 'ImagePath', cdrom)
    cfg.set("Info", 'Volume', cdcfile)
    dirs = walker.keys()
    i = 0
    for d in dirs:
        i+=1
        cfg.set("Comment", str(i),d)
```

```
21    for p in walker:
22       cfg.add_section(p)
23       for f in walker[p][1]:
24          cfg.set(p, f, os.stat("%s/%s"%(p,f)).st_size)
25    cfg.write(open(cdcfile,"w"))
```

有几个地方要注意一下。改进后的函式第 10 行，应该理解为：

- 使用字典保存目录结构信息；
- 以根目录作为 key（键），对应子目录加文件作为 value（值）；
- 为了便于后续的处理，可以根据路径信息，自然地获得进一步的文件信息！并输出到文件的对应数据节中。

done! 当然还有其他连带的改动：

1. 命令行相关响应的实际处理更新。
2. 搜索的相应升级。

这些都是小改动了，随手就得。

小结

在本日故事中，小白通过自发的反省，对比了数据结构的优劣，选择了自个儿最容易理解，且跨平台通用的数据格式，照例在行者提醒后发觉并使用内建模块 `ConfigParser` 来进行数据文本的组织和解读。至少可以体验到：

1. 文本也是数据，结构化的文本不仅仅可以用来作配置文件，也完全可以作为专门的数据库。
2. os 模块包含了在操作系统中文件的所有主要操作。

练习

1. 编程实现以下功能，并进行最大化的优化：遍历指定目录下的所有文件，找出其中占用空间最大的前 3 个文件。
2. 利用 ConfigParser，将上述题目中产生的结果按照 cdays+1-my.ini 格式存储到文件 cdays+1-result.txt 中。

PCS208

PCS208 “ dict4ini ”实际上是针对 .ini 格式文本数据的，行者曾经自定义了相关模块，进行更加 OOP 的操作!dict4ini 就是其中一种第三方模块,在 PCS208 中有相关描述，有兴趣的可以深入阅读。

CDay+2　界面!不应该是难事儿

用户界面友好化

网站软件化绝对不是空话!

清点需求

小白开始自豪地清点到目前为止已经自主实现的功能。

1. 扫描光盘内容并存储为硬盘上的文本文件。
 1） 存储成*.cdc 的文本文件，采用函式 `cdctools.cdWalker()`；
 2） 可以快速指定保存目录，采用函式 `PyCDC.do_dir()`；
 3） 可以快速指定保存的文件名，采用函式 `PyCDC.do_walk()`。
2. 根据储存到硬盘上的光盘信息文件进行搜索。
 1） 可以搜索指定目录中所有*.cdc 文件，采用函式 `cdctools.cdcGrep()`；
 2） 可以指定关键字进行搜索，采用函式 `PyCDC.do_find()`。
 - 列出所有含有关键字的信息行。

使用 GUI 有必要吗?没必要吗

“能不能给个界面啊？每次都要输入命令，很麻烦的……”

呜乎，果然有这种主流反馈汹涌而来了!

小白一边腹诽，一边也在憧憬：“不习惯使用命令行的都是初级用户！咳，俺也喜欢图形界面的……但是好难的!”不用去列表询问，小白也知道 Python 无所不能，一搜索就知

道至少有：

1. `Tk`
2. `wxPython`
3. `Qt`

三大 GUI 框架可以快速组织桌面软件界面出来，但是，就算是写出来了，也无法简单地发布成友好的坚固的稳定的傻瓜化的安装程序呀！

在列表中一问才知道，虽然有 N 多发布打包框架，但是都至少得包含完整的 Python 虚拟机环境，仅这一项就是 5MB，想来核心代码不超过 200 行的东西，发布成过 MB 的东西，小白就感觉脸红……这不白玩 Python 了？一点也不 Cool！

咋办咧？倒个苦水先，在社区列表中行者轻轻一句话惊醒梦里人！

“浏览器也是 GUI！”

“浏览器也是 GUI”

现在不都流行网站软件化吗？ HTML+浏览器，可不就是用户最熟悉，而且最有宽容心理的界面吗？！

注意：SaaS（Software-as-a-Service，软件即服务），官方网站：*http://www.saas.com.cn/*，指出了当前网站的运营服务化、功能软件化的趋势。

好啦！就整个本地站点式的界面出来，要求用户先安装好 Python 环境，然后，解开压缩包，不用安装，运行一条命令，就可以通过浏览器看到一个本地功能网站来使用 `PyCDC`！

Web 应用框架

有时候选择太多，也是种痛苦……再次感叹一下！小白刚刚一动念头，就搜索到这种页面：

WebFrameworks：*http://wiki.python.org/moin/WebFrameworks*

精巧地址：*http://bit.ly/3jxUSJ*

Python WEB 应用框架纵览：*http://wiki.woodpecker.org.cn/moin/PyWebFrameList*

精巧地址：*http://bit.ly/1U4oHm*

密密麻麻一大片应用框架……

“俺不想弄出个复杂的强大的有 DB 支持等高级特性的网站，俺就是想用 Python 快速完成一个有限功能的本地网站呀！”小白在列表中吼道。

很快，行者给出了回应：“Karrigell”。

Karrigell

最简单和省心的 Web 应用框架，就属 Karrigell 了，实在是最听话的一种。

好吧，那就尝试使用 Karrigell 来整个桌面网站型的界面！

PCS301 “ Karrigell ” 精要地介绍了这一 Web 应用框架的主要特性。 PCS301

介绍

http://karrigell.sourceforge.net/ 主站的介绍也非常精妙。Karrigell 为了大家不同的习惯居然提供了多达 5 种 Web 应用发布形式：

PCS300 “ CherryPy ” 另外介绍了这一历史悠久、思想独到的 Web 应用框架。 PCS300

1. 自然 Python 脚本

```
1 #hallo.py
2 print "<h1>Squares</h1>"
3 for i in range(10):
4     print "%s :<b>%s</b>" %(i,i*i)
```

——有大量的 HTML 要亲自组织和输出？不直爽！

2. HTML 嵌 Python 脚本

```
#hallo.hip
"<h1>Squares</h1>"
for i in range(10):
    "%s :<b>%s</b>" %(i,i*i)
```

——同样得亲自组织 HTML 输出？不直爽！

3. Python 嵌 HTML 脚本

```
#hallo.pih
<h1>Squares</h1>
<%
for i in range(10):
    print "%s :<b>%s</b>" %(i,i*i)
%>
```

——这不就是 PHP 吗？没劲！

4. CGI 脚本

```
#hallo.py
print 'Content-type: text/html'
print
print "<h1>Squares</h1>"
for i in range(10):
   print "%s :<b>%s</b>" %(i,i*i)
```

——比自然 Python 还要麻烦，而且对发布目录有约束？不直爽！

5. Karrigell 服务

```
1 #hallo.ks
2 def index():
3    print "<h1>Squares</h1>"
4    for i in range(10):
5       print "%s :<b>%s</b>" %(i,i*i)
```

——看起来好像支持函式化的组合？ 就这种了！

文档 *http://karrigell.sourceforge.net/en/front.htm*（精巧地址：*http://bit.ly/2z3ejE*）有进一步的细节介绍，不过小白已经有足够的信心立即使用 Karrigell 来启动自个儿的程序了。

运行

首先得将环境运行起来，如果可以边修改，边在页面中看到结果的话，就非常直爽了！

也不用问，看着简单的文档，一般就 3 板斧：

1. 下载，安装。

 耶！Karrigell 居然是零安装，超级绿色的！解开压缩包，在任何地方都可以运行！使用不包含 demo 文档的核心版本 `Karrigell-core-2.3.5.tgz`，解压缩了才 800 多 KB！

2. 配置。

 在 `Karrigell-2.3.5` 目录中找到 `Karrigell.ini`（注意：在 Karrigell 2.4 版本之后，配置文件移动到 `conf` 目录中了），然后找到：

```
...
[Server]
# set the port on which Karrigell will serve requests
# default is 80
# on Unix/Linux, if you are not logged as root you may not be able
```

```
    # to start on a port below 1024 for security reasons
    #port=8081
```

仅仅修订一行：

```
    port=8081
```

去掉注释，声明将网站发布到 `:8081` 端口，避开用户可能已有的默认`:80`网站。

3. 运行 python Karrigell.py。

完事儿！ 网站已经启动，界面已经拥有，输出的状态信息如图 CDay2-1 所示。

```
~/LovelyPython/CDays/cday2/Karrigell-2.3.5$ python Karrigell.py
Karrigell 2.3.5 running on port 8081
Debug level 1
Press Ctrl+C to stop
127.0.0.1 - - [20/Sep/2008:18:08:34 +0800] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [20/Sep/2008:18:08:46 +0800] "GET /cdc/ HTTP/1.1" 302 -
127.0.0.1 - - [20/Sep/2008:18:08:46 +0800] "GET /cdc/index.ks/index HTTP/1.1" 20
0 -
```

图 CDay2-1 Karrigell 启动后输出的状态信息

组织

前后看了看，发觉默认的 Karrigell 发布 `Karrigell-2.3.5/webapps` 目录为网站根目录。

为了独立组织各种已有代码，在其中创建 `cdc` 子目录，先再将原先的 `index.html` 复制一份到 `cdc` 中，然后修订默认首页 `index.html`，追加一链接：

```
    <h1><a href="cdc/">PyCDC Web</a></h1>
```

刷新，如图 CDay2-2 所示：

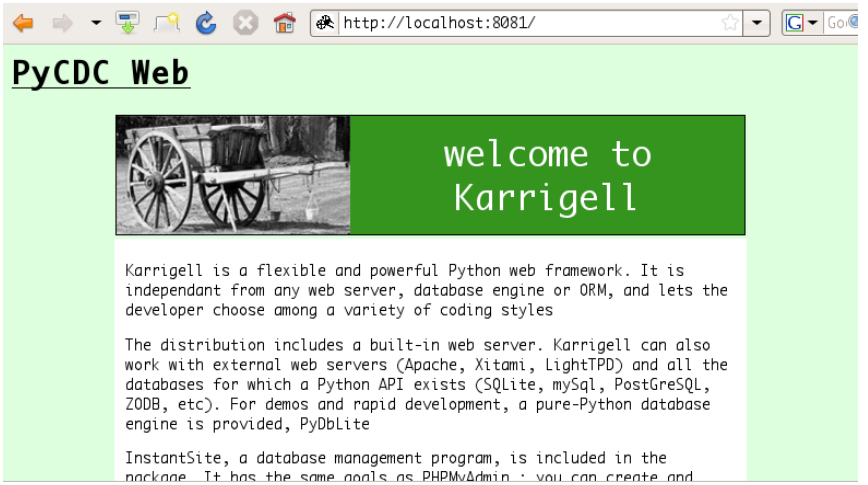


图 CDay2-2 PyCDC 网站首页

呵呵，网页老实地显示出来，而且可以正常点击！

然后就是想法儿将 `cdc/index.html` 重构为 `cdc/index.kss` ，并可以调用原先 `cdctools.py` 中准备好的函式，能在命令行环境中干同样的活就得！

重构

先来个 20 多行：

```
 1 # -*- coding: utf-8 -*-
 2 def _htmhead(title):
 3     '''默认页面头声明
 4     @note: 为了复用，特别的组织成独立函式,根据 Karrigell 非页面访问约定，函式名称
            前加"_"
 5     @param title: 页面标题信息
 6     @return: 标准的 HTML 代码
 7     '''
 8     htm = """<html><HEAD>
 9 <meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
10 <title>%s</title></HEAD>
11 <body>"""%title
12     return htm
13 ## 默认页面尾声明
14 htmfoot="""
15 <h5>design by:<a href="mailto:Zoom.Quiet@gmail.com">Zoom.Quiet</a>
16     powered by : <a href="http://python.org">Python</a> +
17  <a href="http://karrigell.sourceforge.net"> KARRIGELL 2.3.5</a>
18 </h5>
19 </body></html>"""
20
21 def index(**args):
22     print _htmhead("PyCDC WEB")
23     print htmfoot
```

刷新，如图 CDay2-3 所示：



图 CDay2-3　PyCDC 网站 ks 实现

真爽！小白参考网络中搜索到的前辈的实例，一试即成！感觉真不是一般的爽快！而且

有两个小的技巧到手：

PCS109“ 系统参数 ”深入地分享了在 Python 代码中可以声明的各种参数形式，以及适用情景。有兴趣的读者可以深入体验一下 Python 这方面的 Pythonic。

1. 在 Karrigell 中，函式名称有前缀"_" 的话，就会当作私密函式，不认为是可访问的页面，凡是辅助性函式都可以这么来。
2. 直接使用 Python 的函式规范将 `index()` 的参数声明成动态参数`**args`，这样一来将所有功能集成到首页中，当进行点击提交等操作时，`index()` 函式就可以智能地接受不定长度、形式、格式的参数了！

导入

好了，将原先的工具脚本导入进来，就可以通过网页调用原先的功能了，但是，怎样才能从 "CDays/cday2/Karrigell-2.3.5/webapps/cdc/index.ks" import "CDays/cday2/cdctools.py" 中复用已经成熟的功能呢？好像 `import` 操作不像网页的包含动作有相对路径可以玩的？先不管了，复制过来一个，`import`！咦？ 怎么看似导入成功了？

```
from cdctools import *
print dir()    #通过检查名称空间进行测试，结果如图 CDay2-4 所示。
```



图 CDay2-4　import 测试结果

瞧见了几个熟悉的函式名称，DONE！

PS：小白万分感动地看到 Karrigell 提供了友好、丰富、明晰的网页调试信息汇报（如图 CDay2-5 所示）。

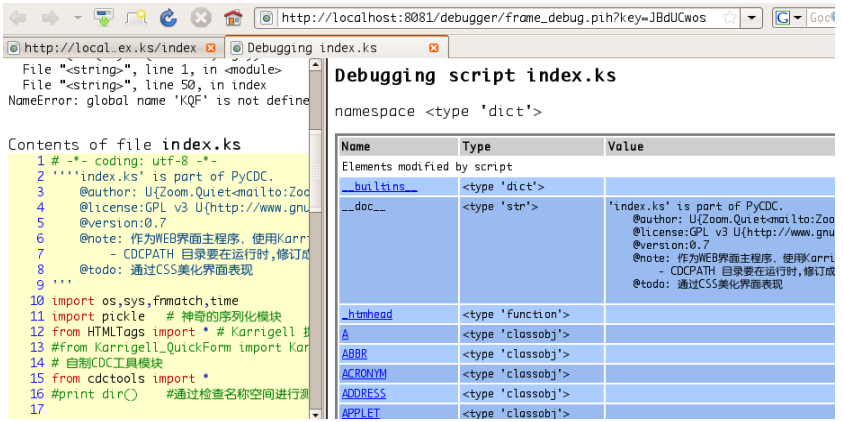


图 CDay2-5　网页输出调试信息

表单

好了，组织好了，接下来就是表单组织，最后连接上既有功能就得！

HTMLTags：*http://karrigell.sourceforge.net/en/htmltags.htm*（精巧地址：*http://bit.ly/fdz7w*）有说，Karrigell 提供一个非常 OOP 的支持模块，可以快速地像函式样儿地组织生成 HTML 代码：

```
    print HTML( HEAD(TITLE('Test document')) +
          BODY(H1('This is a test document')+
              TEXT('First line')+BR()+
              TEXT('Second line')))
```

将生成：

```
      <HTML>
      <HEAD>
      <TITLE>Test document</TITLE>
      </HEAD>
      <BODY>
      <H1>This is a test document</H1>
      First line
      <BR>
      Second line
      </BODY>
      </HTML>
```

但是，可以进行交互提交的表单好像无法函式化的生成呀！有点郁闷，小白不想自个儿折腾，根据“`-1 day`”的经验：

你能够碰到的问题，99%的情况下其他人已经遇到过了。所以，最佳的解决方式就是找到那段别人解决了相似问题的代码！

还是问一声：“Karrigell 有表单快速生成模块吗？”

果然有行者反馈：Karrigell_QuickForm，网址为：*http://wiki.woodpecker.org.cn/moin/KwDay3*（精巧地址：*http://bit.ly/3VBSKI*）

FT!! 连名字都一样：**快速表单！**

```
# -*- coding: utf-8 -*-
from Karrigell_QuickForm import Karrigell_QuickForm as KQF
#...

def index(**args):
    '''默认主页
    @note: 使用简单的表单 / 链接操作来完成原有功能的界面化
    @param args: 数组化的不定参数
    @return: 标准的 HTML 页面
    '''
    print _htmhead("PyCDC WEB")
    p = KQF('fm_cdwalk','POST',"index","CD Walk")
    p.addHtmNode('text',"keywd","文件名",{'size':20,'maxlength':50})
    p.addGroup(["submit","btn_submit","Walk it!","btn"])
    p.display()
    p = KQF('fm_cdwalk','POST',"index","CD Walk")
    p.addHtmNode('text',"keywd","关键字",{'size':20,'maxlength':50})
    p.addGroup(["submit","btn_submit","Search It!","btn"])
    p.display()
    print htmfoot
```

得了您呐！追加 8 行，完成两个功能针对性的表单，如图 CDay2-6 所示。

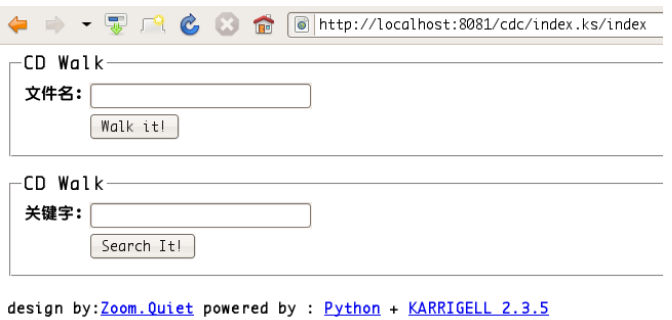


图 CDay2-6　使用 QuickForm 生成页面

搜索功能

随便输入，提交，没有任何问题，但是怎么获得提交的信息？

在网上 *http://karrigell.sourceforge.net/en/programming.htm*（精巧地址：*http://bit.ly/2z6wju*）有说。瞧着这 `QUERY` 内置变量就是亲切啊！

PCS100

PCS100“ import ”进一步说明了模块和包使用的方法和注意事项。

追加：

```
        if 0 == len(QUERY):
            pass
        else:
            print QUERY
        print htmfoot
```

再提交两次，彻底明白了网页参数的传送，绑定原先的功能！

```
    ## 全局变量
    CDCPATH = "/path/to/LovelyPython/CDays/cday2/cdc"
    CDROM = '/media/cdrom0'
    ...
        if 0 == len(QUERY):
            pass
        else:
            if "Search" in QUERY['btn_submit']:
                print I("try search *.cdc for KEY:%s"%QUERY['keywd']),BR(),
                cdcGrep("%s/"%CDCPATH,QUERY['keywd'])
            elif "Walk" in QUERY['btn_submit']:
                iniCDinfo(CDROM,"%s/%s"%(CDCPATH,QUERY['keywd']))
            else:
                pass
        print htmfoot
```

追加 6 行，完成核心功能绑定！

运行结果如图 CDay2-7 所示：



图 CDay2-7　搜索功能演示

成咧!! 呵呵，一切太自然了……

搜索结果“二传”

不过，这搜索匹配结果的输出也忒糙了点儿，随手改进一下吧：

```
# -*- coding: utf-8 -*-

def cdcGrep(cdcpath,keyword):
    export = ""
    filelist = os.listdir(cdcpath)          # 搜索目录中的文件
    for cdc in filelist:                    # 循环文件列表
        if ".cdc" in cdc:
            cdcfile = open(cdcpath+cdc)      # 拼合文件路径，并打开文件
            export += cdc
            for line in cdcfile.readlines(): # 读取文件每一行，并循环
                if keyword in line:          # 判定是否有关键词在行中
                    #print line               # 打印输出
                    export += line
    print export
```

呜乎！依然没有反应，页面输出的居然是 None？！咦？这是什么意思？ 好吧，使出调试的终极招术：**重启服务！**Ctrl+c 键或是直接关闭运行脚本的命令窗口，再次运行 python Karrigell.py。Bingo！ 再次搜索，输出了修订后的结果列表！

小白进一步想了想，在搜索时总是会有重复的搜索，怎么样可以将每次的搜索成果记录下来，以便日后进行分析或是直接复用呢？

那么，**二传**一个！

PCS210

PCS210“pickle”进一步分享了这一常用模块的特性。最可爱的，就是这个模块支持自然的对象到文件对象的双向转换！

其实想用用超级 cool 的模块 pickle（序列化！ 对象腌制器！）据说可以将 Python 的数据对象输出成文本文件，而且导入时直接转化成自然的对象!!!

```
# -*- coding: utf-8 -*-

def cdcGrep(cdcpath,keyword):
    expDict = {}
    filelist = os.listdir(cdcpath)         # 搜索目录中的文件
    for cdc in filelist:                   # 循环文件列表
        if ".cdc" in cdc:
            cdcfile = open(cdcpath+cdc)        # 拼合文件路径，并打开文件
            expDict[cdc]=[]
            for line in cdcfile.readlines():   # 读取文件每一行，并循环
```

```
            if keyword in line:           # 判定是否有关键词在行中
                #print line               # 打印输出
                expDict[cdc].append(line)
    pickle.dump(expDict,open("searched.dump","w"))

if __name__ == '__main__':     # this way the module can be
    '''cdctools 自测响应处理
    '''
    CDROM = '/media/cdrom0'
    CDCPATH = "/path/to/LovelyPython/CDays/cday2/cdc/"
    ## 需要根据实际情况指向真实的目录
    cdcGrep(CDCPATH,"EVA")
    print pickle.load(open("searched.dump"))
```

cdctools.py 运行结果如图 CDay2-8 所示。

```
~/LovelyPython/CDays/cday2/Karrigell-2.3.5/webapps/cdc$ python cdctools.py
{'mfj-00.cdc': [], 'z.Animation.00.cdc': ['[L:\\mAnimi\\Gainax\\EVAalbumESP]\r\n
'], 'MCollec.39.cdc': [], 'z.MFC.pop.02.cdc': [], 'z.MCollection.29.cdc': [], 'm
ymusic-1.cdc': []}
```

图 CDay2-8　cdctools.py 运行结果

对应地导出文件 `searched.dump`：

```
(dp0
S'z.MCollection.29.cdc'
p1
(lp2
sS'mfj-00.cdc'
p3
(lp4
sS'MCollec.39.cdc'
p5
(lp6
sS'z.Animation.00.cdc'
p7
(lp8
S'[L:\\mAnimi\\Gainax\\EVAalbumESP]\r\n'
p9
asS'z.MFC.pop.02.cdc'
p10
(lp11
s.
```

果真可以从纯文本直接变成字典对象呀，Cool！

```
#完善首页代码为:...
```

```
def index(**args):
    ...
    if 0 == len(QUERY):
        pass
    else:
        if "Search" in QUERY['btn_submit']:
            if "" == QUERY['keywd']:
                print H3("pls import SearchKey!")
            else:
                print I("try search *.cdc for KEY:%s"%QUERY['keywd'])
                print BR()#,cdcGrep("%s/"%CDCPATH,QUERY['keywd'])
                cdcGrep("%s/"%CDCPATH,QUERY['keywd'])
                searcheDict = pickle.load(open("searched.dump"))
                for cdc in searcheDict.keys():
                    print H5(cdc)
                    for line in searcheDict[cdc]:
                        print BR(line)
        elif "Walk" in QUERY['btn_submit']:
            print I("try walk CD for:%s ..."%QUERY['keywd'])
            iniCDinfo(CDROM,"%s/%s"%(CDCPATH,QUERY['keywd']))
            print BR(),B("had export info. file as:%s/%s"%(CDCPATH,QUERY['keywd']))
        else:
            pass
    print htmfoot
```

搜索时信息显示如图 CDay2-9 所示：

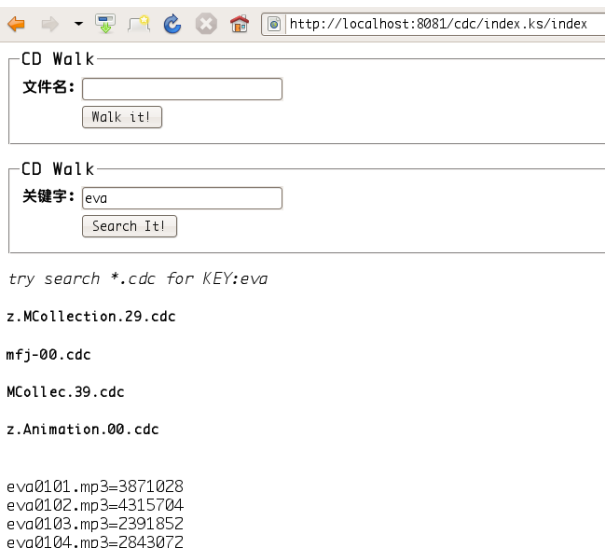


图 CDay2-9　搜索时信息显示

记录时信息显示如图 CDay2-10 所示：

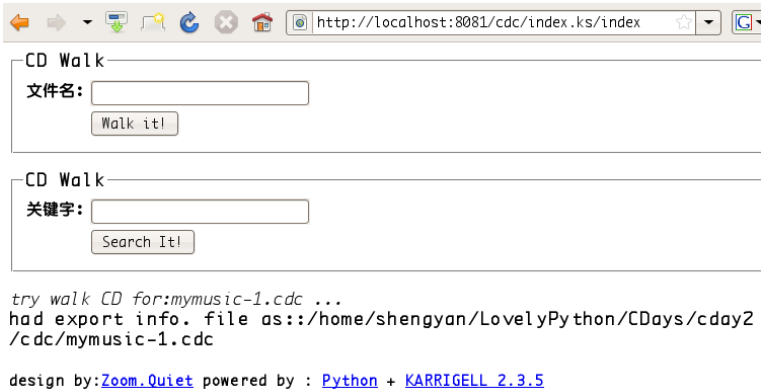


图 CDay2-10　记录时信息显示

TODO

Karrigell 这么听话，几乎是随想随写就完成了 Web 化的界面实现，接下来？

美化！

呵呵，这在 HTML 中比在 GUI 的各种框架代码中可以要 Easy 多了！ CSS 而已。

web.py

web.py 是绝对简单直接的应用框架，但是同样拥有各种高级特性！

核心文件就两个完整的最精简 Web 应用框架，虽然现在也发展到有十来个文件，但是也包含了从 DB 到模板到 Web 服务林林总总的所有高级 Web 应用特性。

不过坚持了最 Pythonic 的直接表述原则，小白看到完整功能网站实例就 8 行：

```
1 import web
2 urls = ('/(.*)', 'hello')
3 class hello:
4     def GET(self, name):
5         i = web.input(times=1)
6         if not name: name = 'world'
7         for c in xrange(int(i.times)): print 'Hello,', name+'!'
8 if __name__ == "__main__":
9     web.run(urls, globals())
```

直接调用 `python tryweb.py`，如图 CDay2-11 所示。

```
~/LovelyPython/CDays/cday2$ python tryweb.py
http://0.0.0.0:8080/
127.0.0.1:57396 - - [23/Sep/2008 15:43:17] "HTTP/1.1 GET /" - 200 OK
127.0.0.1:57396 - - [23/Sep/2008 15:43:18] "HTTP/1.1 GET /favicon.ico" - 200 OK
127.0.0.1:57396 - - [23/Sep/2008 15:43:21] "HTTP/1.1 GET /favicon.ico" - 200 OK
127.0.0.1:58129 - - [23/Sep/2008 15:43:39] "HTTP/1.1 GET /a" - 200 OK
127.0.0.1:58129 - - [23/Sep/2008 15:43:46] "HTTP/1.1 GET /hehe" - 200 OK
```

图 CDay2-11　web.py 框架使用初探

就会在本地 8080 端口发布一个可以响应用户 URL 输入的网站！不过，实在太简单了，什么都要自个儿来，不如 Karrigell 做的舒服呢！

小结

在本日故事中，小白通过响应其它小白的新界面需求，探索了一下 Web 应用框架，照例在行者的提点下开始了首个轻量级框架 `Karrigell` 的使用；至少可以体验到：

1. 在桌面，本地网站是可以视作用户软件界面的！
2. 一个可用的 Web 应用框架，至少应该包含直观的模板系统，方便的自动更新 / 调试支持。
3. 永远有辅助性的第 3 方模块可以加速我们想法的实现，不过，要付出努力去寻找。

到现在，小白已经无须学习什么新的 Python 语法点了，一切已经足够小白所关注需求的快速实现，甚至于可以直接阅读他人的代码来理解既有模块的功能了……Python 的易学、易用、易读的特性可见一斑！

练习

1. 如果在 Karrigell 实例中，不复制 cdctools.py 到 webapps 目录中，也可以令 index.ks 引用到？
2. 经过本章对 Karrigell 的初步学习，实现一个简易的 Web 留言系统。主要利用 Karrigell_QuickForm 实现提交留言并显示出来。
3. 思考本日提出的搜索结果积累的想法，如何实现？ 如何在搜索时可以快速确认以前曾经搜索过相同的关键词，直接输出原先的搜索成果，不用真正打开 CD 信息文件来匹配？

CDay+3　优化!多线程

应用多线程再次优化

KISS 才是王道!

盘点功能

小白又在自满地清点着到目前为止已经自主实现的所有功能了。

1. 一个基于命令行的界面，可以：
 1） 将光盘内容索引存储为硬盘上的文本文件。
 - 存储成*.cdc 的文本文件；
 - 可以快速指定存储目录；
 - 可以快速指定文件名。
 2） 根据储存到硬盘上的光盘信息进行搜索。
 - 可以根据指定关键字进行匹配搜索；
 - 可以搜索指定目录中所有*.cdc 文件。
2. 一个基于 Web 网页的界面，同样可以：
 1） 将光盘内容索引存储为硬盘上的文本文件。
 - 存储成*.cdc 的文本文件；
 - 可以快速指定文件名。
 2） 根据储存到硬盘上的光盘信息进行搜索。
 - 可以根据指定关键字进行匹配搜索；
 - 可以搜索指定目录中所有*.cdc 文件。

已经超额完成了当初想要实现的所有功能了！但是，人生不如意者，十之八九也！

为什么这么慢

在小白兴高采烈地将过往百多张 VCD，几十张 DVD 都列成 `*.cdc` 摘要文件通过 PyCDC 管理起来后，发觉，在搜索时真的非常的慢啊！

列表中也有人在吼！“俺就是将硬盘备份 DVD 导入了一下，10 万左右个文件，5000 左右的目录，怎么搜索起来这么慢啊！”

这问题就严重了，小白当然不认为是 Python 天生慢，看着就几行相关的代码，他也知道是自个儿的功力不够，根本没有想过执行效率的问题。

好吧，该优化时就优化，没有什么不好意思的！

不过，咋整呢？

分析

经过不断的重构，反思，小白已经非常习惯先分析，再找现成代码来偷懒的“开发”模式了：

原先对搜索速度没有感觉，现在却感觉很慢，唯一变化的是文件数量和总体积。

影响速度的行为应该有：

1. 打开文件
2. 文件读取
3. ini 解析
4. 搜索匹配
5. 输出

以上元素，除了搜索匹配，其他都是使用内置模块实现的，基本没有效率问题。但是，搜索匹配也是简单匹配，几乎没有什么好改进的啊！唉呀呀！晕！小白突然想到，整个流程中，是按照过滤出的文件名进行的，每一个文件都要走上述所有过程后才轮到下一个，可以说强大的计算机几乎一直在等待简单地输入输出后，再处理下一个文件！

并发处理

让电脑同时搜索多个文件不就好了么？！

进程 Vs 线程

但是！并发什么来处理？

小白也道听途说过什么进程和线程的，那么使用什么方式好呢？没有头绪呢，吼吧！

呵呵，看来是吼到点子上了，列表中行者们的回复丰富得很！

“进程是操作系统可以调配的最小资源集合！”

“Python 支持多线程的，干什么不用？”

“线程才是操作系统可以调配的最小资源单位！”

“一个程序至少有一个进程，一个进程至少有一个线程。”

“一个线程可以创建和撤销另一个线程，同一个进程中的多个线程之间可以并发执行。但是多个进程中的不同线程很难交互数据……”

晕了，彻底晕了，这都在说些什么呀？ 小白自个儿参照着做了点研究，确认：

1. 进程和线程就是程序执行时请求的系统不同级别资源的名称；
2. 多进程就像孙悟空猴毛变的多个分身，大家自个儿干自个儿的，牵扯较少；
3. 多线程就像哪吒的三头六臂，一个人同时干多个事儿，还能相互配合。

那当然是多线程了！毕竟搜索结果是得集成在一起输出汇报的呀！

KISS

KISS == Keep It Simple,Stupid

在列表中，小白记住了 `KISS` 原则，不论什么，坚持简单的过程/原理/结构/代码，就是自在！所以，参考模块手册（ thread-objects），找个实例代码：Python 线程编程的两种方式（作者：guanjh）。

照猫画虎准没大错!

PCS206 “ thread ”详细说明了线程支持模块的基本使用方法。

实现

PCS207 “ threading ” 详细分享了高级线程模块的基础应用。

用 threading 模块的类组织实现，比用 thread 模块中的线程函式更加容易掺入不同的处理过程，就用 threading 模块了！

按例改造 cdctools.py，追加准备进行并发搜索的线程对象：

```
# -*- coding: utf-8 -*-
from threading import Thread                # 导入线程支持模块
class grepIt(Thread):
    def __init__ (self,cdcfile,keyword):
        Thread.__init__(self)
        self.cdcf = cdcfile
        self.keyw = keyword.upper()
        self.report = ""
    def run(self):
        if ".cdc" in self.cdcf:
            self.report = markIni(self.cdcf,self.keyw)
```

呵呵，忒简单了，就几行，而且，实际行为就一个！markIni() 也是改造自原先的 cdcGrep()。

```
# -*- coding: utf-8 -*-
def markIni(cdcfile,keyword):
    '''配置文件模式匹配函式:
    '''
    report = ""
    keyw = keyword.upper()
    cfg = rcp()
    cfg.read(cdcfile)
    nodelist = cfg.sections()
    nodelist.remove("Comment")
    nodelist.remove("Info")
    for node in nodelist:
        if keyw in node.upper():
            print node
            report += "\n %s"%node # error as "\n",node|str(node)...
            continue
        else:
            for item in cfg.items(node):
                if keyw in item[0].upper():
```

```
                report += "\n %s/%s "%(node,item)
    return report
```

在末尾追加实际搜索调用，并声明成模块测试：

```
# -*- coding: utf-8 -*-
def grpSearch(cdcpath,keyword):
    '''多线程群体搜索函式:
    '''
    begin = time.time()
    filelist = os.listdir(cdcpath)         # 搜索目录中的文件
    searchlist = []                       # 记录发起的搜索编程
    for cdcf in filelist:
        pathcdcf = "%s/%s"%(cdcpath,cdcf)
        current = grepIt(pathcdcf,keyword)  # 初始化线程对象
        searchlist.append(current)         # 追加记录线程队列
        current.start()                   # 发动线程处理
    for searcher in searchlist:
        searcher.join()
        print "Search from ",searcher.cdcf," out ",searcher.report
    print "usage %s s"%(time.time()-begin)
if __name__ == '__main__':
    '''直接利用模块测试部分,先检验是否正确可用
    '''
    grpSearch("cdc/","EVE")
```

这样，就可以直接在命令行中运行 `python cdctools.py`，测试其是否可用了，而且根据看到过的代码随手追加了运行时间汇报，可以输出搜索总用时，运行结果如图 CDay3-1 所示：

```
~/LovelyPython/CDays/cday3$ python cdctools.py
/media/cdrom0/NamieAmuro/neverEnd/音乐极限--日韩专区.files
/media/cdrom0/NamieAmuro/neverEnd
Search from  cdc//z.MFC.pop.02.cdc  out
Search from  cdc//MCollec.39.cdc  out
 L:\可儿乐队精选The Corrs Greathits/('i never loved you anyway.mp3', '3281377')
 L:\可儿乐队精选The Corrs Greathits/('everybody hurts.mp3', '4899889')
Search from  cdc//z.Animation.00.cdc  out
Search from  cdc//mfj-00.cdc  out
 /media/cdrom0/NamieAmuro/neverEnd/音乐极限--日韩专区.files
 /media/cdrom0/NamieAmuro/neverEnd
Search from  cdc//.svn  out
Search from  cdc//z.MCollection.29.cdc  out
 L:\MFW.20\jazz\Info\Mp3之家-绝版JazzCD.files/('apelevencoveraa.jpeg', '5650')
 L:\MFJ.16\The Brilliant Green/('uptu.com-rainy_days_never_stays03.wma', '926441
')
 L:\MFJ.16\The Brilliant Green/('uptu.com-rainy_days_never_stays02.wma', '807552
')
usage 0.135001897812 s
```

图 CDay3-1　cdctools.py 改进后运行结果

TODO

啊！主体功能摸索完毕！然后呢？然后就顺势而为了：

1. 修订命令行界面的代码配合；
2. 修订 Karrigell 的 Web 桌面界面的代码配合；

进一步优化速度还可以怎么整呢？缓存 ini 的对象化解析?! 不用每次每个.cdc 文件都重新解析一回。缓存搜索过的关键字结果?! 有相同的搜索就不用再来一遍了。缓存……

看起来可以做的还有很多，而且，可以想到这些，小白已经不是小白了！可喜可贺!

小结

在本日故事中，小白由实现大规模使用原创工具的真实体验中引发了又一轮重构。针对执行效率的重构，就像玩头脑急转弯，只要充分直觉地分析运行时的整体软件行为，很容易确定瓶颈。

对问题有了准确的理解后，再向行者请教时便有了确切方向，进而容易获得有效的提示；不过，的确有些电脑的基础 FAQ 是回避不了的，在理解了进程和线程后，Python 内置的模块就可以解决了：

1. threading 模块是个易用好用的线程支持模块，可以快捷地将合适的行为并发化。
2. 多线程的调试一定要设计有合理明确的输出信息，以便确认哪里有问题。

练习

1. 熟悉线程相关知识后，利用 Lock 和 RLock 实现线程间的简单同步，使得 10 个线程对同一共享变量进行递增操作，使用加锁机制保证变量结果的正确。
2. 使用 Queue 实现多线程间的同步。比如说，10 个输入线程从终端输入字符串，另 10 个输出线程依次获取字符串并输出到屏幕。
3. Python 中的 Event 是用于线程间的相互通信的，主要利用信号量机制。修改题 1 的程序，利用信号量重新实现多线程对同一共享变量进行递增的操作。

CDayN　基于 Python 的无尽探索

想象力才是 Pythoner 的唯一界限！

需求的遐想

从最初的需求到今天，小白已经独立完成了至少以下功能。

1. 一个基于命令行的界面，可以：

 1） 将光盘内容索引存储为硬盘上的文本文件。

 - 存储成*.cdc 的文本文件；
 - 可以快速指定文件名。

 2） 根据储存到硬盘上的光盘信息进行搜索。

 - 可以根据指定关键字进行匹配搜索；
 - 可以搜索指定目录中所有*.cdc 文件。

 3） 同时为每个摘要文件发出独立线程来并发地进行关键字搜索。

2. 一个基于 Web 网页的界面，同样可以：

 1） 将光盘内容索引存储为硬盘上的文本文件。

 - 存储成*.cdc 的文本文件；
 - 可以快速指定文件名。

2） 根据储存到硬盘上的光盘信息进行搜索。

- 可以根据指定关键字进行匹配搜索；
- 可以搜索指定目录中所有*.cdc 文件。

3） 同时为每个摘要文件发出独立线程来并发地进行关键字搜索。

所有需求都在 50 行代码以内就实现了核心功能，小白感到非常欣慰，经过清点代码，真正地对 Python 树立起了足够的自信！

他这才对社区里 Python 简介中的开场诗有了全面的理解：

左咖啡，右宝石；还是灵蟒最贴心！

最贴心，不费心，用好只须听故事。

想清楚，就清楚，一切自己来动手！

要清爽，常重构！刚刚够用是王道！

然后结合各方信息开始尝试。

啄木鸟社区专精于 Python 技术在中国的应用和推广，收集有大量相关资料，也有专用的推广文章：http ://wiki.woodpecker.org.cn/moin/SpreadPython（精巧地址：http：//bit.ly/16xO0W），其中的“ Python 简介 ”(http://www.zoomquiet.org/share/s5/intropy/070322-introPy/精巧地址：http：// bit.ly/3suMWb）是常用的“ 忽悠 ”Python 的入门手册。

桌面化

PCS217 “ Tkinter ”中的 Tkinter 是 Python 中一种比较流行的图形编程接口。Tkinter 模块是 Python 的标准 Tk GUI 工具包的接口。TK 和 Tkinter 是为数不多的跨平台的脚本图形界面接口。

PCS217

一定要启动个 Web 服务器才能看到漂亮的界面吗？一定要忍受简单的字符命令行界面吗？自然是可以进化的！Python 可以使用的 GUI 框架也不少：

- Tk/Tcl
- Qt
- WxPython
- win32com

PCS302 “ Leo ”作为纯 Python 实现的轻巧文学化编辑环境，已经演化成了文学化应用平台，各种插件层出不穷，Leo 本质上只是 Tk 的一个跨平台文本编辑软件。但是由于其独特的文本处理视角，引发了各种应用，这里快速分享了 Leo 的世界观，引导读者进一步探寻文学化编辑的世界。

PCS302

还有其他一些框架，甚至可以使用 mozilla 框架的 XUL 接口来利用 FireFox 的界面生成软件，只须选择一个简单有趣的来尝试就好。

FP 化

一定要 OOP 开发吗？

现在的 CDC 仅仅是接受光盘信息和查询关键字，如果从面向数据的开发考虑，完全可

以跃迁成基于 FP 的函式型程序。一切都是函式间的嵌套调用，一旦发生问题就会自然退出，无模式，无状态，只是单纯的数据响应，整个结构可以更加扁平不易出错，好维护。Python 同样内置有一些标准的 FP 算子：

- lamba()
- map()
- zip()
- apply()
- reduce()
- filter()

更加魔幻的是在 Python 2.4 以后追加了“修饰符”（@）。可以通过更加简单的方式进行函式叠加了， Python 有能力将自个儿写成 Lisp 一样的纯函式程序，不过，除了 Cool 一时想不出有什么具体的好处。

PCS114

PCS114 “ FP 初体验 ”进一步说明了内建的一些函式化算子的使用，帮助读者体验什么是 FP~函式化编程，这是种古老而且高效易维护的编程思路，虽然和主流的命令序列式编程体验完全不同，但是真的是非常值得学习和理解的好东西。

C 化

对速度和内存有洁癖的人，是否可以使用 Python 来快速开发，却可以得到 C 样的最高效执行效能？

Pyrex 是一种用来编写 Python 扩展模块的语言。简单地说，就是可以使用 Python 来完成功能，然后通过 Pyrex 生成同样功能的 C 代码，以供编译生成最高效执行程序！

而且也有其他各种可以渗入 C/C++的支持模块：

- ctype
- boost
- cxx
- WarpPy
- SWIG
- SIP

想 C 化 Python 应用也只是几分钟的事儿！

Flash 化

Flash 可以说是现在最漂亮的界面创造方式了， Python 应用可以使用 Flash 作前端吗？

随便看了看 Flash 的 ActionScript 脚本，发现非常像 JS，而且可以通过 XML 进行交互，好的 Python 的 XML 解析和处理也不是摆设。

完全可以通过 Flash 提供美丽动态的界面，由 AC 组织好事务 XML，丢给 Python 处理实际数据，然后依然是 XML 返回，达成软件的可用性。

只是，这其中还是需要个标准的 Web 服务器提供给 AC 进行 URL 访问，好在 Python 自行创立个简单的 Web 服务也实在不算什么，利用现成的 Karrigell/web.py 等超轻量级 Web 应用框架也一样轻松。

分布化

好的，如果我们的光盘多到无法在一台机器保存所有摘要信息，或是需要有不同的服务器来分类保管，或是需要同步收集 DVD 信息，Python 是否也可以支持？一查才知道，哗！世界第一的 BT 下载体系——bittorrent 根本就是 Python 开发的！而且类似 EVE 的大型网络游戏也是使用 Stackless Python 进行开发的。那么，分布的话也应该没有问题，关键是分布到什么程度。

小白期望：

- 可以自动将不同主机上的实例收集的 DVD 摘要信息相互同步到本地。
- 可以自动分发搜索要求，并将各自的结果汇总到任何一台发出查询请求的主机处。

压缩化

现在的摘要信息是以 .ini 的格式存储在文本文件中的，如果多了有些浪费空间，是否可以像其他软件那样弄个自己的压缩格式？

Python 内置支持从 .zip 文档中导入数据对象？！

好吧，不用努力就可以获得的特性，soooooo easy！

定制化

操作的命令，或是页面上的元素、输出的方式，是否每个人都可以自行调节呢？

这和 Python 没有直接关系，是 CSS、Ajax 的事儿了，不过，只要是通过 XML 等标准数据沟通的模式，Python 就是没有问题的！

智能化

现在的查询实在忒简单了点，我要像 Google 那样的组合条件！要有正则表达式的模糊模式匹配！呵呵，正则表达式是内置在 Python 中的，组合条件不过是参数的解析，不应该是个问题吧？

Google 化

想协助地球上所有其他有光盘管理需求的人们使用 PyCDC 进行轻便地光盘仓库管理？使用 GAE （Google App. Engine ）服务！

- 透过 Google 的免费应用发布环境！
- 透过 Google 的强力搜索引擎服务！
- 使用 Google 的轻型分布式数据仓库！
- 借助 Google 的无限空间邮箱服务！
- 组合 Google 的其他各种免费强力服务！

将 PyCDC 变成依托 Google 公司的全球化服务吧！唯一的要求是你会 Python。

PCS400

PCS400 “ GAE ”简要地介绍了 2008 年 4 月刚刚发布的 Google App. Engine 服务，分享了在 GAE 中快速建立自个儿原创应用的方式／方法／思路。

我化

还想将小白的学习成果 PyCDC 怎么折腾？读者自个儿来吧，没有人可以比你更加理解你的需要，深入学习 Python“我化”这一小工具好了。

想象力才是 `Pythoner` 的唯一界限！

小结

充分信任 Python，大胆设想，小心求证，搜寻已有实例，谁都可以创造出美妙实用的工具！

KDays “ Web 应用故事 ”

KDay0　Web 开发启航　82
KDay1　品尝 KarriGell　86
KDay2　通过表单直接完成功能　93
KDay3　使用第 3 方模块规范化表单　101
KDay4　使用 KS 模式重构代码　114
KDay5　通过 session 重构应用流程　123
KDay6　利用 mm 人性化组织成员信息　135
KDayN　经验总结，畅想 Web 应用　147

KDay0　Web 开发启航

定场诗

前樱桃，后涡轮，还是推车最贴心。（CherryPy TurboGears KarriGell）

最贴心，不省心，一切头绪要理清。

想清楚，就清楚，一切清楚才清爽！

要清爽，常重构！才有更爽在后头！

以上是小白通过了 CDays 的历练后从某位行者那儿瞧到的打油诗，拿来作这篇实例故事的开场白。这是因为，在 Python 的帮助下，实现了 CDCollector 的大部分功能之后，小白真切地感受到了这首诗所传达出的那种感慨。

缘起

在工作学习之余，小白努力想将 Pythonic 的体验分享给同事和同学们，以至上司和老师们都知道有个 Python 脚本语言可以快速完成各种任务。所以，一有事他们首先就想叫小白来尝试一下。这不，小白接到了个比较正式的开发任务——改建一个简单的在线问卷系统，可参考现已使用的一个简单的在线问卷。

小白看了一下源代码，感觉非常简单，就自发地将任务转变成**开发一个简单的在线问卷管理系统**（暂定名为“EasyPaper）”，可以方便地创建简单的问卷页面出来。

小白主动地将这次基于 KarriGell 的 Web 应用开发过程记录成了实例故事，以连载的方式分享给社区。

不过，小白自知不是什么写书能手，只是想分享 Python 开发的快捷体验，没有经验和自信能与没有任何 Pythonic 体验的初学者产生共鸣，所以，小白设定了故事准入条件——想看接下来的实例故事的人，须满足以下条件：

1. 了解互联网；
2. 了解 HTML；
3. 了解 CSS；
4. 了解 DHTML；
5. 了解动态页面的含义；
6. 了解 PHP 或是 ASP 的开发思路；
7. 了解 Python；
8. 了解 CherryPy。

PCS300 PCS300“ CherryPy ”分享了樱桃蟒 这一最早出现的关注 OOP(面向对象编程)体验的应用框架，基础使用体验。

所谓了解，就是对以上方面技术:明了相关基础知识，看过/写过相关代码，知道具体是什么性质和范畴的知识；否则，读者一定会在第一时间感到困惑，进而倍感挫败的。

另外，本故事面向那些控制欲强——对于任何系统，如果不是所有代码都是自个儿写的，会感觉非常不靠谱，不敢使用的那种人，他们可能会比较喜欢这篇故事，否则，对所谓“成熟框架类”体系非常崇尚的，期望在大量优雅的自动生成的框架代码帮助下快速完成开发的人，一定会感觉这样开发太别扭。

PCS401 PCS401“ DHTML ”分享了动态 HTML 这一常用网络应用技术的知识点，以及连带涉及 PHP/AS 等动态网页技术。

故事

角色

小白：

已经不是菜鸟的小白，以第三人称的方式，记述了自个儿在实例开发活动中的各种体验。

行者：

啄木鸟/CPyUG/ZEUUX 等等中国活跃技术社区的那群热心的 Python 用户，在 Python 应用/学习方面是先行者，但都不是专业教师，所以，说话可能有些颠三倒四，但都是真心助人的好人。

约定

1. 每节故事都提供可运行的 KarriGell 实例站点脚本，以便通过实际运行，得到直观的体验。

 注意：所有实例代码使用 SVN 提供下载，访问 URL 地址为：

 http://openbookproject.googlecode.com/svn/trunk/LovelyPython/KDays/

 精巧地址：*http://bit.ly/2aAaUy*

 SVN 是 Subversion 的简称，是种非常流行和稳定的版本管理系统；简单讲，就是一个中央服务器及配套工具集；可以方便地管理所有开发时的各种文件；神奇的是，它可以帮助你安全地取回任何时间点上的文件/目录——即使文件/目录曾经被删除过。进一步的介绍公布在 *http://wiki.woodpecker.org.cn/moin/SubVerSion* 上。
 精巧地址为：*http://bit.ly/b7w9K*

PCS302 “Leo”中指出 Leo 是种极具个性的文学化编辑环境，使用独特的方式来组织我们的软件工程，在 PCS302 分享了初步体验。

2. 实例故事，不是教程，小白只会讲述要点，具体的都含在脚本代码中了，读者只要下载运行，尝试按照自个儿的理解修订一下，看一看运行的反应，就可以深入到系统的各个层面，理解其细微之处。
3. 实例脚本全部使用 Leo 组织！小白也推荐读者可以尝试通过 Leo 来观察代码，这样可以感受到小白组织代码时的思路，也可轻松掌握代码的整体框架。
4. 每日故事涉及的开发，基本上都是在 3.1415 小时之内就可以完成的。
5. 每日故事最后的小结，是小白养成的好习惯：“及时清点成果或是问题，同时给明日的开发定出可行的目标。”
6. 每日故事最后的练习，是小白根据故事涉及的知识点和领域技术所设计的一些问题，列出来是期望读者独立进行尝试，以强化相关体验。

 习题解答发布在图书维基：

 http://wiki.woodpecker.org.cn/moin/ObpLovelyPython/LpyAttAnswerKdays

 精巧地址：*http://bit.ly/axi7*

 用 SVN 下载：

 http://openbookproject.googlecode.com/svn/trunk/LovelyPython/exercise/part2-KDays/

目标

小白最终的开发目标，就是使“EasyPaper”工具能作到以下功能：

- 可以通过有格式约定的纯文本生成问卷；
- 可以批量生成问卷；
- 生成的问卷可以立即受理登录和回答；
- 问卷的回答可以得到实时的成绩统计；
- 可以随时修订格式约定的纯文本，从而变更已经发布的问卷。

接下来给大家看一个实例问卷定义文本样例。

```
#easy051201.cfg
[desc]
pname          = 啄木鸟问卷 之 “基本知晓”
desc           = 自学问卷 v0.7
learn   = <a href='http://wiki.woodpecker.org.cn/moin/CPUG'>CPUG 首页</a>
# 问卷状态: 0 设计中|1 发布中|2 发布过
done           = 0

[ask/1]
question= 啄木鸟社区首页在哪里？
a              = woodpecker.org.cn
b              = python.cn
c              = 不知道……
key            = a # 正确答案
[ask/2]
……
```

提示：文本支持注释，约定凡是 # 开头的行，就略去，不用作问卷生成。

练习

尝试运行 Karrigell，其下载地址为：*http://sourceforge.net/projects/karrigell*（精巧地址：*http://bit.ly/1i7NyI*）。

- 下载 Karrigell 解压；
- 在终端中进入 Karrigell 所在目录；
- 运行 `python Karrigell.py`，看看出现什么?
- 再在浏览器中输入 localhost，再看看出现什么?

如果出现意外，尝试根据屏幕提示进行排除。

KDay1　品尝 KarriGell

快速根据现成的 PHP 实现思路，尝试 Karrigell 的开发

背景

一般的问卷系统其实就是一组手工写成的非常简单的 PHP 页面，每个页面包含一个选择题，点击“下一题”就可以将每个人对应的回答记录到数据库中，其中：

- 每页头部包含前页题目的正确回答；
- 由 PHP 通过 URL 参数判定回答是否正确，并将前一题的回答情况记录到数据库中；
- 然后通过数据库的查询进行成绩统计。

小白学习了 Python 后，首先写了个脚本，可以从 .ini 文本中自动批量生成相似的 PHP 页面：

- 每个类似 `[ask/1]` 领头的一节内容包含一系列键值对，对应生成一个选择题及其答案；
- 其中在 `key=` 一项记录了正确答案的选择项。

然后配套也写了自动查询 DB，统计出回答问卷的成员成绩的 Python 脚本，所以，现在的问卷生产流程是:

```
.ini(txt) -> Python -> PHP -> DB -> Python -> 答卷统计
```

这样的数据流程真够傻的，经过了太多手续，使用了太多的语言和中间处理，而且无法作到答卷成绩的立等可取！

PHP（Personal Hypertext Preprocesser，个人超文本解析）语言是种专门针对互联网应用设计的快速功能型语言，对于网站开发有独到的处理思路，但是不属于本书分享的范畴，请读者自行搜集相关文档学习，TisonG！……

准备环境

总算得空，小白立即想尝试使用纯 Python 重构出一个可快速修改+统计成绩的简单问卷系统。不过，为什么选择 KarriGell 呢？

小白想这么回答：“缘分吧……呵呵！”真实原因就是“懒”，毕竟在前述 CDays 实例故事中，在 Python 学习过程中，小白已体验过了 KarriGell，当然，就首先使用熟悉了，哈哈哈！

立即开始！

下载 KarriGell，一般是个单独的压缩文件，比如说 `Karrigell-2.4.0.tar.bz2`。

下载地址：*http://sourceforge.net/projects/karrigell*

精巧地址：*http://bit.ly/1i7NyI*

接着，使用工具解开压缩（推荐使用 7-zip 这一自由软件，通吃一切常见压缩包！ 网址为：*http://7-zip.org/zh-cn/*）。

然后，进入 KarriGell 所在目录， 将 conf/Karrigell.ini 中的#port=8081 去掉注释，表示将网站发布到 `:8081` 端口，避开用户可能的已有默认`:80` 网站。

最后，在命令行下面输入：`python Karrigell.py`。M$用户的命令行环境调用：开始→命令→输入“cmd”（或是你习惯的任何 M$终端软件）；Unix/Linux 用户则调用：应用程序→附件→终端（或是其他任何你喜欢的终端软件）。

bingo！通过 *http://localhost:8081/* 小白就已经获得了一个展示有各种应用的 KarriGell 站点了！其默认首页如图 KDay1-1 所示。

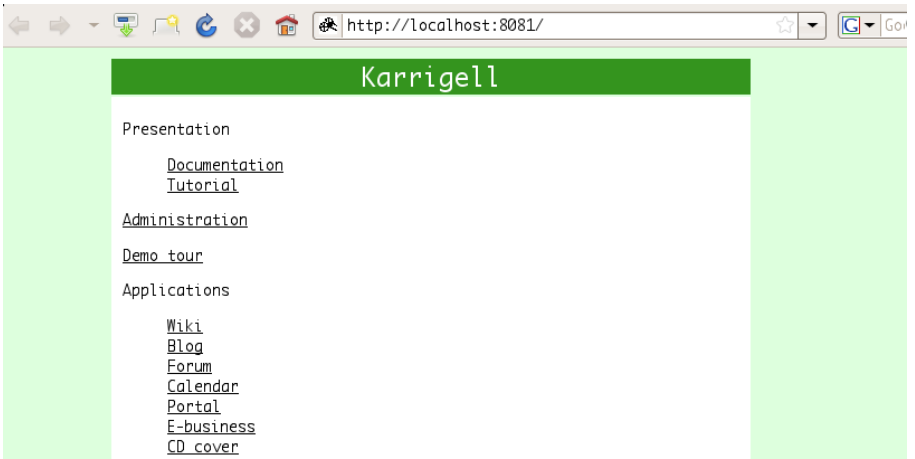


图 KDay1-1　默认首页

PCS404

PCS404“ 代码重构浅说 ”中谈到的重构（Refactoring）是现代软件开发工程中常用技巧，可以保证系统在可用状态下持续不断地提高代码品质！在 PCS404 中，浅显地说明了一下重构思想及常见的重构技巧。

PCS301

PCS301 “ Karrigell ” 分享了这一被选中的号称最简易好用的应用框架的基本配置和开发使用体验;就在本书前一部分，CDays 故事中，最后小白就是使用的这种框架。

这也意味着，只要有 Python 环境，我们可以在任何平台中，随时获得一个全功能的 Web 应用环境！

动手动脚

跟着教程走，随便动一动，在默认的 KarriGell 站点中，就含有教程（如图 KDay1-2 所示）！

http://localhost:8081/demo/frame_tour_zh.htm 是最简要的入门教程，展示了 KarriGell 进行 Web 开发中最经常要面对的应用实现。

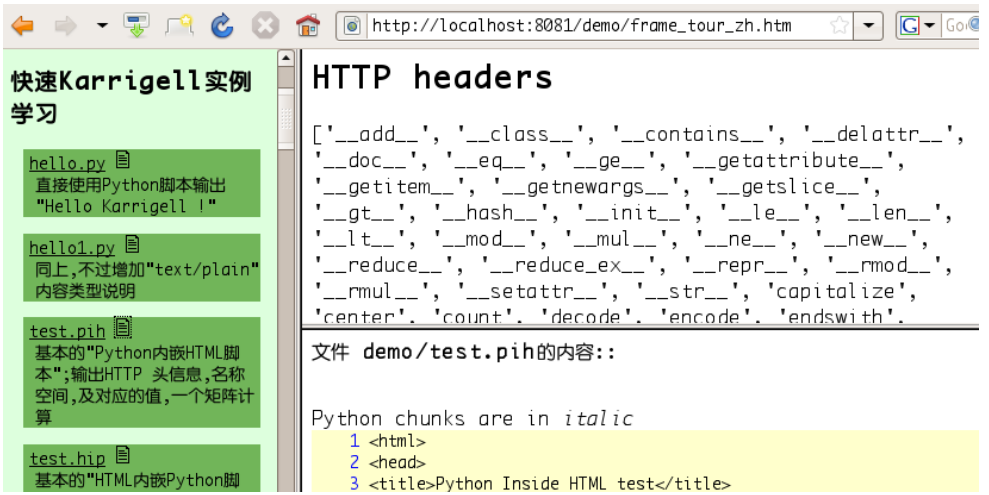


图 KDay1-2　内置中文教程

小白也算修炼过 Python 编程的，当然对于“hello world”这种无用代码已没有兴趣了，想直接进行交互式的 Web 应用开发。经过行者的点拨，注意到有种 `.pih` 发布方式，类似原先的 PHP 页面，可以将执行代码嵌入 HTML 页面中。

根据内置教程的演示，小白直接对原有页面 *http://localhost:8081/demo/test.pih* 进行了修改:

```
<h2>HTTP headers</h2>
<%
a="Karrigell"
print dir(a)
%>
```

呵呵！！页面输出（如图 Kday1-3 所示）果然和命令行中的反应一样，看来就将 KarriGell 想象为面向 Web 的 Python 解释器好了！

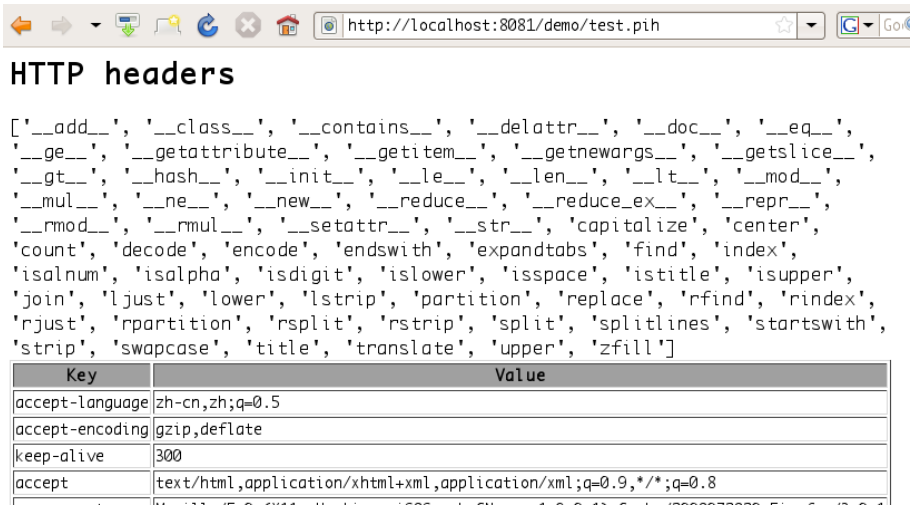


图 KDay1-3　打印 dir()的页面输出

组织开发环境

“OK！我期望可以随时看到教程，同时又有独立的站点空间……”小白这样想着的时候，机缘巧合下学习了 Leo ，于是他认为一切都应该由 Leo 组织才自然。那么就按照习惯来定制 KarriGell 吧！

其实 KarriGell 的设置是非常简单的，由统一的 ini 文件来设定整体行为。我们现在的需求非常简单，就是将 KarriGell 的默认教程站点和小白自个儿的开发隔离开，以便开发的同时不影响随时参考内置教程，所以查找有关目录别名的配置……我们在 ini 文件找到以下代码。

```
[Alias]
...
# demo=%(base)s/demo
doc=%(base)s/doc
debugger=%(base)s/debugger
```

Yeah！就是这样，非常直观，那么定义一个自个儿开发的站点目录在 KarriGell 系统目录中就好。呵呵，这里要感谢 Unix 系统中的 link 命令！小白可以将开发中的问卷系统文件放在任何习惯的目录中，仅仅链接到 KarriGell 的环境目录中就完成了自个儿的应用安装；如果在 M$ Windows 环境中就只能在 KarriGell 的环境目录中建立目录了。

比如说，在 Linux 中，小白的 EasyPage 工程组织在：

```
~/LovelyPython/KDays
```

而下载安装的 Karrigell 在:

```
~/0penproj.s/trunk/karrigell/trunk
```

则小白只要运行:

```
$ ln -s ~/LovelyPython/KDays/kday1 \
  ~/0penproj.s/trunk/karrigell/trunk/webapps/
```

就等于在 ~/0penproj.s/trunk/karrigell/trunk/webapps/ 中安装好了~/LovelyPython/KDays 中的 kday1 目录。然而若在 M$ Windows 环境中，小白就只有将 kday1 目录复制到 KarriGell 的环境目录下了。这样，小白想灵活地将不同目录中的不同工程目录配置到不同版本的 Karrigell 实例环境中，是无能为力的。

PCS302 “ Leo ” 分享了什么是文学化编辑环境，以及利用文学化视角来组织软件工程时的最佳体验;这里的 @nosent 是指 Leo 中结点的特殊名称，可以引起不同系统响应的名称。 PCS302

首先，习惯性地在 Leo 中设立好自个儿的工程，组织一个最简单的 index.pih：

```
<html><body>
<%
a="Kariigell"
print dir(a)
%>
</body></html>
```

为了方便，使用 Leo 在节点声明 `@nosent` 操作符，以便按下 `Ctrl+S` 时，就自动在指定的目录里保存相应文件（如图 KDay1-4 所示）。



图 KDay1-4　Leo 界面中的情景

对应地，在配置文件 Karrigell.ini 中设定一下。

```
[Alias]
...
k=%(base)s/webapps/KDays
...
```

OK！重启一下 KarriGell 。这里试用一下 Linux/Unix 系统中的通用操作 `Ctrl+C`，令 KarriGell 的 Python 进程停止，然后再运行 `python Karrigell.py`。其后访问地址: *http://localhost:8081/k*，哈哈，一切如愿！KarriGell 正如小白想象的有了实时的反应，不像 CherryPy 每次修改，都要重启服务。

这是因为在 KarriGell 中，就像 PHP 开发那样，每次修改好后，刷一下页面，你的修改立即就能显示在页面输出可以和服务运行对比，如图 KDay1-5 所示。

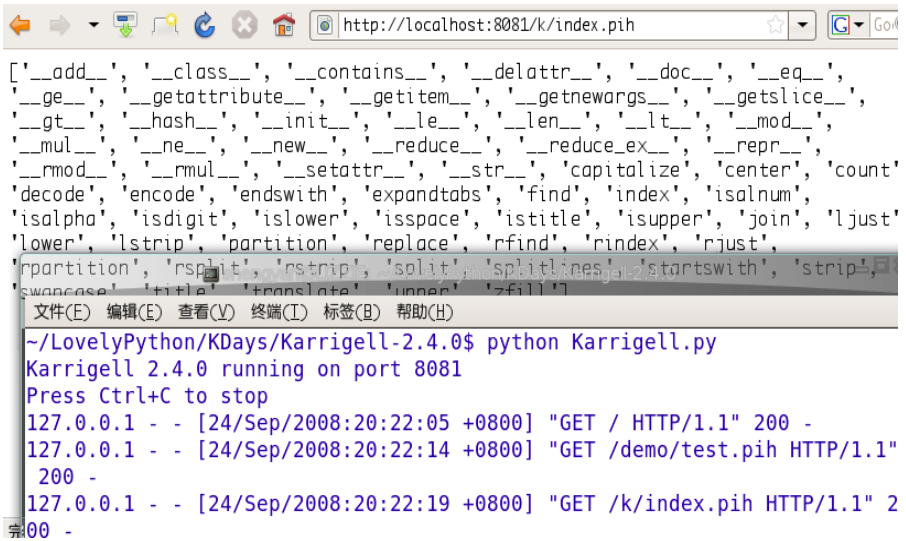


图 KDay1-5　页面输出和服务运行对比

小技巧

```
[Directories]
...
allow_directory_listing = all
```

在实际开发过程中，经常要查看实际发布目录中有什么内容，打开配置文件中的上述配置项，Karrigell 将自动发布没有默认首页的目录内容索引，其默认的目录内容索引样式如图 KDay1-6 所示。

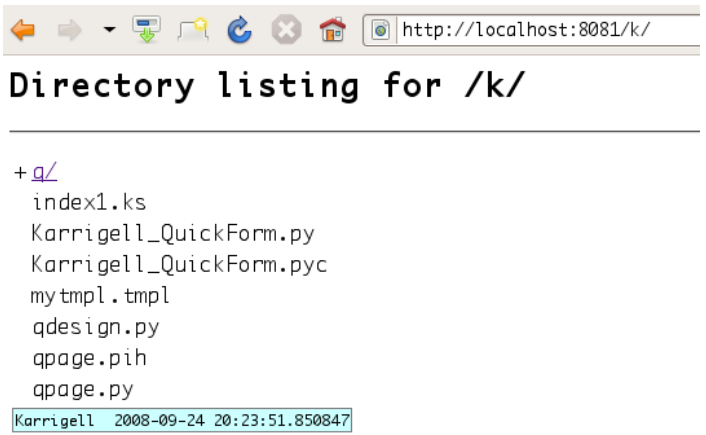


图 KDay1-6　默认的目录内容索引样式

小结

简单回想一下今天任务的完成情况：

PCS208 “dict4ini”分享了专门用以解析和输出 .ini 类 M$ Windows 配置文件文本的支持模块的基本使用思路;这是 UliPad 作者 Limodou 自个儿组织分享出来的。

1. 评估 KarriGell 的可用性——已经达成！
2. 简单地确定 KarriGell 的应用开发/组织方式——已经达成！

那么，接下来确定明日计划。

1. 对于“简单问卷”本身，先要完成的是怎么在 KarriGell 中读入外部文件，并可以在页面中修改，然后提交保存……
2. 利用大侠 Limodou 自创的 Dict4Ini 模块来理解问卷的设计，并将问答收集为一系列文件，最终仍由 Python 统计成绩……

实例下载

实例下载时请使用 SVN 下载地址。

实例下载: *http://openbookproject.googlecode.com/svn/trunk/LovelyPython/KDays/kday1*
精巧地址：*http://bit.ly/4qP7S8*

我们来回顾一下本日成果：

```
kday1/
|-- q/                      问卷设计文本收藏目录
|   `-- easy051201.cfg      第一份问卷设计
`-- try.pih                 第一个动态页面
```

练习

经过这一天的学习， 熟悉 Karrigell.ini 基本设置和基本环境部署后（可参考 README.txt 中），创建一个自己的站点（mysite），并依次实现以下功能:

- 将 Karrigell 站配置发布到 localhost:8081 端口；
- 在首页（index.pih）中输出“Hello，Karrigell's world！”即在浏览器的地址栏中输入 *localhost:8081/mysite* 页面显示出“Hello，Karrigell's world！”
- 添加一个要求有密码控制的登录页（login.pih），即当输 *localhost:8081/mysite* 后，首先进行登录，我们输入姓名和密码后，转入首页，其上显示“Hello，姓名's world！”具体可参见相关截图。

KDay2　通过表单直接完成功能

不管三七二十一直接完成心目中的功能!

小白秉承之前的习惯，先计划当日期望达到的目标：

- 什么可以直接读取原先的问卷设计.ini 文本，并显示到页面中的文本框（<TEXTAREA></TEXTAREA>）中；
- 什么可以提交包含问卷设计文本框的表单，并将内容保存到服务器上的指定目录中，完成问卷的更新。

规划

行者 Rockety 对 KarriGell 的设置有很好的介绍。小白参考了相关的介绍后重新规划了站点的开发，并对/path/to/Karrigell/conf/Karrigell. ini 相关部分进行修订。

```
...
[Alias]
..
q=%(base)s/KDays
```

追加这一行设置行，这样一来，就可以通过 *http://localhost:8081/q/kday2/*来访问今天的开发成果了！但是没有在 obp 目录中安置 index.htm/index.pih 之类的默认首页，KarriGell 是不会让小白看到什么的！

约定以下全局性变量：

```
qpath = "q/"
pubq = qpath+"easy051201.cfg"
```

Rockety 的 Karrigell 使用体验，访问地址：http://wiki.woodpecker.org.cn/moin/RocketyKarrigell
精巧地址：
http://bit.ly/3wEwn3
另外在 PCS301 中也对其最常用部分的配置进行了说明。

使用这样的全局变量定义下来，使用 Leo 快速地将所有的文件控制起来，这样计划就进一步明确了：

- 在页面 *http://localhost:8081/q/kday2/mana.pih* 看到并可以编辑 ini 问卷设计文稿，点击提交后转到；
- 在页面 *http://localhost:8081/q/kday2/qpage.pih* 可以看到真正保存下来的问卷设计内容。

Cheetah

小白很早之前从身边的师兄师姐那儿就模糊地听说过 MVC 模式，说什么将数据模式/前端表现/业务控制等分层实现；但是在这个简单得连数据库都没有的实例小项目中，顶多是 VC 分层了，在技术列表咨询了一下，有行者推荐说模板系统中 Cheetah 非常地好和稳定，那就先尝试它了！

稍稍看一下子示例，就可了解到 Cheetah 的基本使用是这样的:

```
1 from Cheetah.Template import Template       # 0. 引入模板
2 page = open("你的模板文件.tmpl","r").read()    # 1. 加载模板
3 vPool = {'cfgtxt':"随便什么字串的值就成"}         # 2. 加载数据
4 print Template(page, searchList=[vPool])    # 3. 渲染输出
```

- 模板文件中有 $cfgtxt 的地方就会替换为实际数据；
- 模板文件就是标准的 HTML 文件，不同之处在于各个期望有动态数据出现的地方，变成了 **$开头的变量名**；
- **$开头的变量名** 只要在合适的时机，先于 print Template() 完成赋值就好。

以如下模板文件片段为例：

```
<!--你的模板文件.tmpl；
```

其实模板文件可以使用任何后缀名的，甚至是.exe，只要内容是纯文本的，但是为了统一，建议还是使用一看就知道是模板的后缀。

```
-->
<H2>$cfgtxt</H2>
```

PCS304

PCS304 “Python Web 应用框架纵论”综合列举了流行的一些 Web 应用框架系统，同时也分享了一些实用的模板选择和使用体验。有关 Cheetch 模板系统的简要介绍在:
http://wiki.woodpecker.org.cn/moin/CheetahTemplateOrg
精巧地址：
http://bit.ly/31kKBK。

Leo 组织实现

小白已经习惯了通过 Leo 在一个统一界面中把握程序的全部逻辑层次/章节/元素。

可以使用中文作章节名，通过**问卷**来组织设计文案，就是@nosent easy051201.cfg。当

然要使用`@path q` 来配合，这样你对 easy051201.cfg 的修改可以立即输出为具体的文件。

`Ctrl+Alt+c` 和 `Ctrl+Alt+v` 是 Leo 中的复制和粘贴操作快捷键。快速从原先的 `@nosent index.pih` 复制整个节点为：

- `@nosent mana.pih` 管理页面入口
- `@nosent questionnaire.tmpl` 修改问卷模板，Cheetah 的。

VC 分层

至此，小白组织好了自个儿的 VC 实现：`@nosent mana.pih` 虽然几乎是纯 HTML 页面，但是通过`<%Include("qdesign.py")%>` 来包含一个纯操作脚本，我们把它看做数据控制（Control）层，`@nosent questionnaire.tmpl` 模板，我们把它看做表现（View）层。

哦，原来所谓分层实现，就是将自然的做法起个 NB 的名称。

八股文样

模式化的处理脚本

从文学化编程角度看，Web 应用的前端应用脚本，应该说都一个样儿！图 KDay2-1 是使用 Leo 规格化的功能页面设计情景。



图 KDay2-1　使用 Leo 规格化的功能页面设计情景

即：

PCS302 “ Leo ” 分享了什么是文学化编程环境，以及利用文学化视角来组织软件工程时的最佳体验。

PCS302

1. 脚本说明 `@...@c` 部分；
2. 脚本声明 `<< page declarations >>` 引用的下层部分；
3. 行为定义 @others 包含的所有下级节点；
4. 实际尝试 <<try>> 引用的下层部分。

编辑实现

编辑实现其实就是将指定的文件内容读出来发布到页面的输入框（<TEXTARE/>）中。我们在模板中先做准备：

```
<textarea NAME="cfgfile" rows="27" >
$cfgtxt
</textarea>
```

然后处理脚本：

```
1 #简化引用对象名
2 from Cheetah.Template import Template as ctTpl
3 vPool = {}
4 vPool['cfgtxt'] = open(pubq,"r").read()
5 page = open("questionnaire.tmpl","r").read()
6 txp = ctTpl(page, searchList=[vPool])
7 print txp
```

完成！如图 KDay2-2 所示。



图 KDay2-2　问卷设计主页面

最后运行！

展示实现

展示实现，指的是将 ini 的内容整理为 HTML 页面展示。

同样快速地组织一下:

1. `@nosent qpage.pih` 访问的页面 *http://localhost:8081/q/kday2/qpage.pih*；
2. `@nosent qpage.py` 实际的数据重组。

关键代码

- `From dict4ini import DictIni`：这里使用的是 Limodou 提供的 dict4ini.py（字典化 ini 操作模块）；
- 创建 `qpage.py→def expage(dict)`：问卷输出函式，来将 ini 内容整理为相应的页面。

```
exp +="<ul>"
# 将字串的字典键值依照数字方式排序
k = [int(i) for i in dict.ask.keys()]
k.sort() # 没有回传的数组重整处理
for i in k:
    ask = dict.ask[str(i)]
    exp +="<li>%s"%ask["question"]
    exp +="<ul>"
    qk = [j for j in ask.keys()]
    qk.sort()
    for q in qk:
        if 1==len(q):
            exp +="<li>%s"%ask[q]
        else:
            pass
    exp +="<p>正确答案::%s</p>"%ask["key"]
    exp +="</ul>"
    exp +="</li>"
exp +="</ul>"
return exp
```

上述代码中的双层循环就可以对应地将所有类似`[ask/1]` 的问题，以及其中的所有类似“`a = 赞！`”的选择项按照列表的形式输出，如图 KDay2-3 所示！

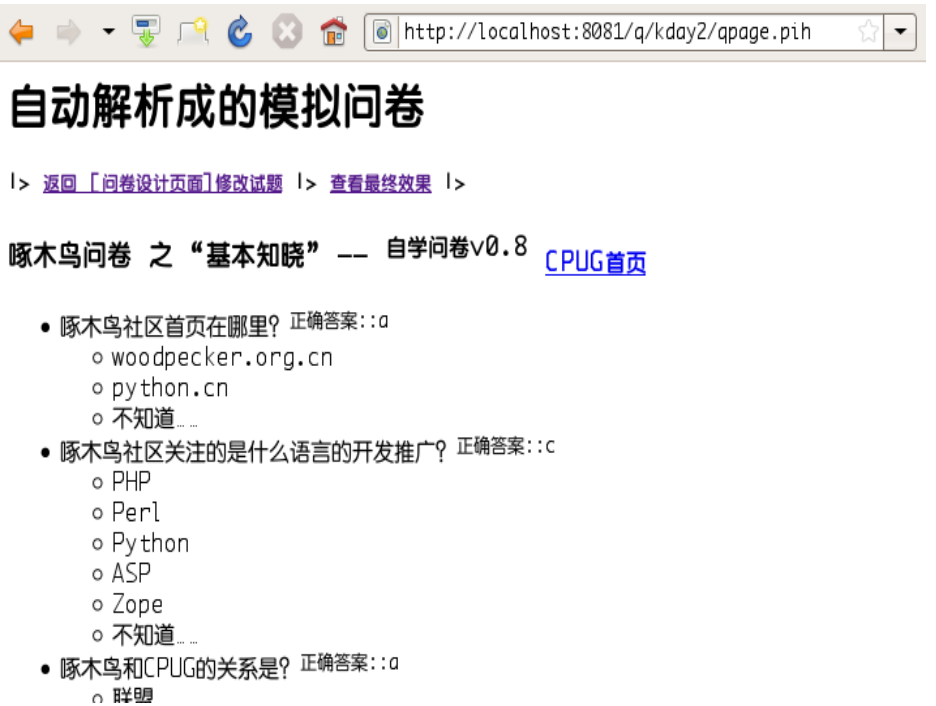


图 KDay2-3　问卷设计文本解析输出的页面效果

PCS208 “dict4ini”知名行者 Limodou 从 UliPad 项目中贡献出来的对象化操作 ini 配置文本支持模块；在 PCS208 中分享了基本技巧。

串联页面

串联页面，即将编辑页面和展现页面串联起来。只要使用 HTML 表单元素（<FORM/>），利用内置的提交声明 `ACTION` 就可以了，例如：

```
<FORM ENCTYPE="multipart/form-data"
    ACTION="qpage.pih"
    METHOD=POST>
```

在模板中补充以上声明。

测试为先！我们在 qpage.py 中加入 `print QUERY`，以确认表单到底传送过来了什么，如图 KDay2-4 所示。



图 KDay2-4　使用 Karrigell 内置对象 QUERY 观察表单提交

OK！一切吻合想象，的确是个字典对象的传送，键值就是 <TEXTAREA/> 元素的 `NAME`；得到问卷设计文本的修订内容后，就可以简单地通过 `open(qpath+pubq, "w").write(QUERY["cfgfile"])`将传送来的编辑成果先写回文件，再由后续操作整理展现，这已经不是问题了。

明日目标

实现了最基本的编辑到保存功能，还有问卷展示的脚本，那么自然地可以想到明天的目标是：实现真实使用的问卷表单！

实例下载

实例下载同样使用 SVN 下载地址。

实例下载：*http://openbookproject.googlecode.com/svn/trunk/LovelyPython/KDays/kday2*

精巧地址：*http://bit.ly/H6sXX*

再来看看本日的成果，如图 KDay2-5 所示：

```
kday2/
|-- dict4ini.py                    Limodou 贡献的 ini 解析模块
|-- q/                             问卷设计文本收藏目录
|   `-- easy051201.cfg             第一份问卷设计
|-- mana.pih                       问卷设计页面
|-- qdesign.py                     问卷设计实际行为脚本
|-- qpage.pih                      问卷展示页面
|-- qpage.py                       问卷展示实际行为脚本
`-- questionnaire.tmpl             问卷展示模板
```

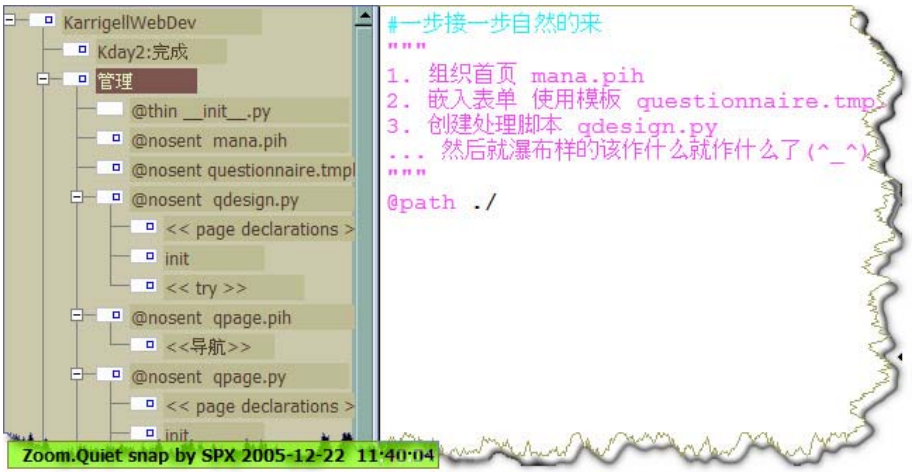


图 KDay2-5　本日完成的所有页面在 Leo 中的组织情景

练习

在前面章节练习 mysite 的基础上，完成以下功能：

- 增加编辑页面（edit.pih），增加编辑输入框，并保存入本地文件，转入首页 index.pih 显示刚刚输入的内容，类似于制作个人博客的编辑页面； 具体页面分布可参考 png 中截图；
- 允许文章的多次修改及删除，并能体现在首页 index.pih 中；

 提示：各个文章可以保存为.cfg 格式，该格式文件的处理可参见 ConfigParser 的使用。

KDay3　使用第 3 方模块规范化表单

Cheetah 只能输出有配套模板的页面内容，但是……

对于经常变化的表单，有什么更好的方式来生成它？

小白开始了对 Pythonic 的重构思考，感觉 VC 格局中的 V —— view 展示层，少不了人手工对 HTML 格式的页面模板进行维护，而且每个对应的功能页面就要写个模板，再者说模板文件的复用也非常困难……所以，想起在 CDays 故事演练中使用过的 Karrigell 快速表单模块 Karrigell_QuickForm。

Karrigell_QuickForm

既有之，则用之！

来看个示例：

```
 1 from Karrigell_QuickForm import Karrigell_QuickForm
 2
 3 p = Karrigell_QuickForm('teste','POST','foo.py','Authentication')
 4 p.addElement('text','login')
 5 p.addElement('text','password')
 6 p.addRule('login','required',"Login is required!")
 7 p.addGroup(["submit","botao_enviar","submit","Send"]
 8          ,["reset","botao_limpar","reset","Clear"])
 9 """根据习惯 hack！原先的自动生成 value 的为指定按钮文字.
10 p.addGroup(["submit","botao_enviar","Send"]
```

```
11          ,["reset","botao_limpar","Clear"])
12 """
13 p.display()
```

6 行完成一个标准的登录表单！！！赞！运行结果如图 KDay3-1 所示。

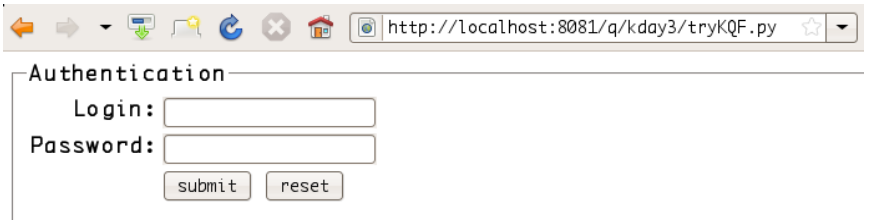


图 KDay3-1　使用 KQF 生成的试验表单

改造

毕竟是 alpha 版本，居然还是全面的 table 结构！代码也很 ugly 呐!（如图 KDay3-2 所示）

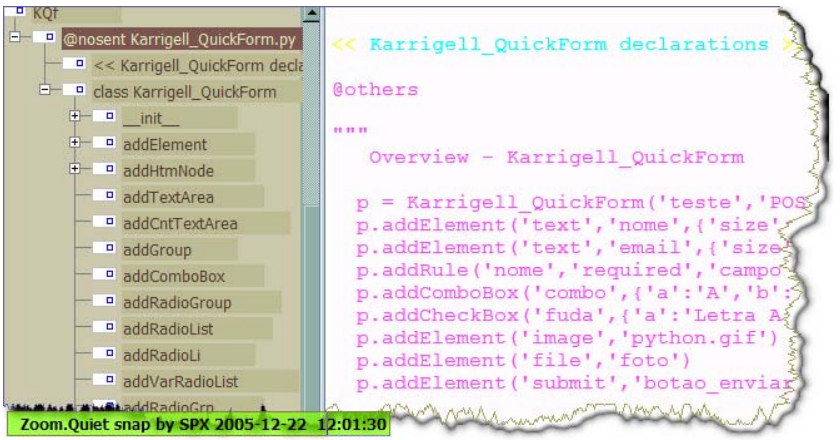


图 KDay3-2　使用 Leo 自动整理后的情景

立即着手改造单选列表生成以下代码：

```
1 def addRadioGroup(self,name,value):
2       """add a radio element - addRadioGroup('genre',{'male':'Male',
        'female':'Female'})"""
3       self.form_list.append("<tr><td align='right' valign='top'><b>"
        +name.title().replace('_',' ')+":</b></td> <td valign='top'
        align='left'>")
4       radio = ""
5       t = value.items()
```

```
6        for a,b in t:
7           radio = radio + "<input type='radio' name='"+name+"' value='"
            +a+"'>"+"<label><font face=verdana size=2>"+b+"</font>
            </label><br>"
8        self.form_list.append(radio+"</td>")
9        self.form_list.append("</tr>")
```

上面的代码是原版的单选列表生成器，从定义的类名称“addRadioGroup”我们可以理解为这段代码的作用是追加单选按钮组，其功能是生成一组单选选项；但郁闷的是，原版代码将单选选项组塞到了一个表格行中，小白认为这样做不妥当，应该是使用列表配合 CSS 来建立单选选项组，这样，HTML 代码更少、控制更加自由，增补一个基于列表的干净版本的单选框生成器：

```
1 def addRadioList(self,name,desc,value,id=""):
2    """add a radio element export as UL LI group
3    """
4    htm = """
5       <li id='%s'><b>%s:</b>
6          <ul>"""
7    self.form_list.append(htm%(id,desc))
8    radio = ""
9    t = value.items()
10   tmpl = """<li>
11      <input type='radio'
12      name='%s'
13      value='%s'>
14      <label>%s</label>
15      """
16   for a,b in t:
17      radio = radio + tmpl%(name,a,b)
18   self.form_list.append(radio+"</ul></li>")
```

利用

再直接将昨天的展示函式修改如下。

def expage(dict): 对应的修改为：

```
1 def qpublish(dict):
2    exp = ""
3    p = Karrigell_QuickForm('fm_kq','POST','#',dict.desc.desc)
4    p.addElement('node','<ul>','')
```

```
    # 深入数据 基本和昨天的一样,仅仅是输出时使用 Karrigell_QuickForm 对象而已
    qli = {}
    k = [int(i) for i in dict.ask.keys()]
    k.sort()
    for i in k:
        ask = dict.ask[str(i)]
        qk = [j for j in ask.keys()]
        qk.sort()
        for q in qk:
            if 1==len(q):
                qli[q] = ask[q]
            else:
                pass
        p.addRadioList("cr_ask%s"%i
                  ,ask["question"]
                  ,qli)
    p.addElement('node','</ul>','')
    p.addGroup(["submit","btn_submit","提交"]
              ,["reset","btn_reset","重写"])
```

注意到原先的 Karrigell_QuickForm 只有 display()，要求表单页面立即输出，但是现在须进一步地经 HTML 处理后再输出，所以有了以下修订。

```
def export(self):
    """ export the html form code so people can do something for them self
    """
    exp = ""
    ...
    for c in self.css_list:
        exp += c+"\n"
    for i in self.form_list:
        exp += i+"\n"
    return exp
```

这个修订基本上就是将原先的 `print` 替换为 `exp+=` 记录为字串对象然后返回。还有在 `def addElement(self，element，name，options=None):` 中追加更自由的任何 HTML 节点输出：

```
         elif element == 'node':
             self.form_list.append(name)
```

得到了什么结果？

JS 问题

一切顺心，表单顺利自动生成了，但是，Karrigell_QuickForm 提供的前端表单检验居然不支持 Radio 列表!

这可是个问题呀！因为，如果不是所有人将所有问题都回答完就提交的话:

- 成绩统计没有意义；
- 交给服务端检验则浪费带宽！还要想办法记录上次是谁回答的问题，然后再返回/提示/要求重答，等等。

继续发掘

现在的问题是：有什么现成的、可以模式化的、定义表单检验的前端 JS 组件?

TiosnG ![ti'aosn'gu]

There is one site named Google!——哈哈！！运用这个流传在行者中的咒语，翻查了一下，确认 `Validation`（有效性检查）是小白想要的检验的学名，于是得到 JVF（JavaScript Validation Framework）——Javascript 有效检验框架，国人作品！如图 KDay3-3 所示。



图 KDay3-3　2004 年当时的 JVF 作品页面情景

PCS401

JVF 是 AMOWA 社区作品，该社区致力于创建一个实用精巧的 Ajax 框架。当年的官网是 http://www.amowa.net，现在迁移到 http://buffalo.sourceforge.net/JVF。
相关的动态网页技术，在 PCS401 DHTML 进行分享。

访问地址: *http://cosoft.org.cn/projects/jsvalidation*
精巧地址: *http://bit.ly/2icRKJ*

迁就，先!

为了与 KarriGell 配合，当前须要:

1. 在配置文件中，追加声明 `jvf=%(base)s/karriweb/questionnaire/js` 这样的专门虚拟目录发布，以便其他各种应用也可以享受 JVF 支持;
2. JVF 实际运行的 validation-framework.js 本身，也要声明可访问的目录：`var ValidationRoot = "/jvf/";`
3. 小量修订 JVF 声明错误输出的页面元素类名 `#errorDiv`;

```
ValidateMethodFactory.validateRequired = function(field, params) {
      ...
      window.location.replace("#errorDiv");
```

4. 使用 replace 来减少不必要的页面刷新;
5. 使用 `p.saveJSRule("../js/validation-config.xml")` 声明，提交时要检验的表单元素。

下面介绍一个测试 JS 引用是否成功的**小技巧**：

```
`alert("Include KO!");`
```

在 JS 脚本中加入强制提示，刷新页面，如果见到已经包含的信息就表明引用路径对了!

KQF 对 JVF 的迁就

JVF 的使用很有个性，须要使用外部的 XML 文件检验行为的设置。所以，须对应增补 Karrigell_QuickForm，步骤如下：

1. addJSRule() 追加专门的 JVF 规则。

```
1    def addJSRule(self,name,message):
2       """add a xml rule for javascript checking
3       """
4       exp = self.JSvMXLnode%(name,message)
5       self.JSvRules.append(exp)
```

2. addJSValidation() 追加调用 JVF 的页面行为。

```
    def addJSValidation(self):
       """add a javascript rule in order to validate a form field
       - addRule('elem_name','required','Name is required!')
       """
       orig = "enctype='multipart/form-data'"
       repl = """
          onsubmit='return doValidate("%s");'
          """
       begin_form=self.form_list[0].replace(orig
                                 ,repl%self.name)
       self.form_list[0] = begin_form
```

3. saveJSRule() 记录规则集合为 JVF 需要的 XML。

```
    def saveJSRule(self,xml):
       """exp and save a xml rule for javascript checking
       """
       exp = ""
       for node in self.JSvRules:
          exp+= node
       #exp = self.JSvXMLtmpl%(form,exp)
       open(xml,'w').write(self.JSvXMLtmpl%(self.name
                                       ,exp)
                                       )
```

4. 对应的 KQF 中追加统一的预设声明。

```
self.JSvXMLtmpl="""<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE validation-config SYSTEM "validation-config.dtd">
<validation-config lang="auto">
       <form id="%s" show-error="errorMessage" onfail=""
       show-type="first">
   %s
       </form>
</validation-config>
"""
self.JSvMXLnode = """
             <field name="%s"
             display-name="%s" onfail="">
                    <depend name="required" />
             </field>
"""
self.JSvRules = []
```

果然不出所料

仅仅追加少量代码就完成了所想的客户端 JS 验证功能。在原先问卷解析函式中，追加准备好的 JVF 支持（如图 KDay3-4 所示）。

```
def qpubish(dict):
    ...
    ## 具体问题解析
    k.sort()
    for i in k:
        ...
        p.addJSRule("cr_ask%s"%i,"问题%s "%i) # 声明此处要进行 JS 检验
    ## 整体行为处理
    p.addJSValidation()
    p.saveJSRule("js/validation-config.xml")    #收集检验声明,生成 JVF 使用的
                                                 外部 XML 设置文件
    ...
```

没有任何悬念地完成任务!



图 KDay3-4　加载 JVF 后,表单有任何项没有提交就报错

页面编码

从开始使用 Karrigell，小白在 FireFox 浏览器中就发现有中文乱码问题（如图 KDay3-5 所示）。



图 KDay3-5　页面乱码现象

小白努力检查了源代码、模板代码，将所有应该声明或是本身已另存为了 utf-8，甚至于发现配置文件中有疑似输出编码声明的地方也逐一打开，比如说:

```
[Server]
...

outputEncoding = utf-8

...

# determine if form fields should be encoded

encodeFormData = 1
...
```

但是依然是乱码，每次非得手工将浏览器的字符编码设置成 utf-8 才正常。这是怎么回事呢？小白不得不到列表中吼，经过行者们的研究，确认是 Karrigell 内部处理编码有问题，不过，想解决也非常简单，使用以下的页面响应编码声明即可：

```
RESPONSE['Content-Type'] = "text/html; charset=utf-8"
```

行者们讨论中文乱码问题的原始记要在啄木鸟社区维基:
http://wiki.woodpecker.org.cn/moin/MiscItems/2008-08-05
精巧地址:
http://bit.ly/2y9Y33

合理使用 Leo

到现在页面都是白板!不能容忍了!使用 CSS!

小白先不理会 KarriGell 的外部文件引用效率，决定直接将 CSS 写入页面，反正有 Leo 来维护（如图 KDay3-6 所示）。

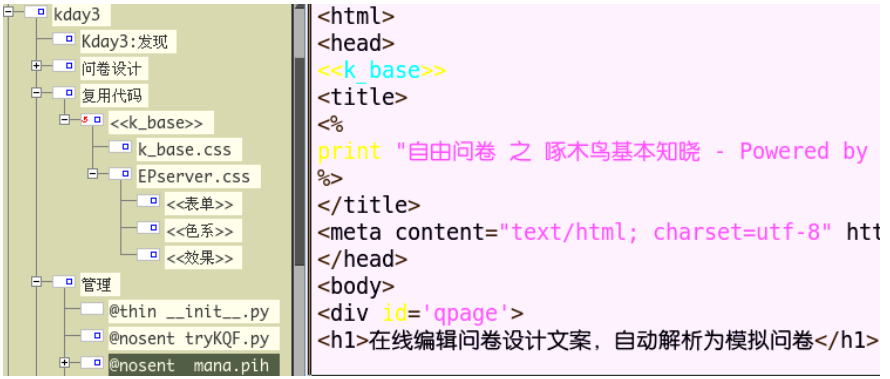


图 KDay3-6　通过 Leo 每个页面复用嵌入的 css 定义

注意 <<k_base>> 节点前的箭头记号，这表明此节点是从别处克隆过来的!

CSS 设计技巧

复用以前自个儿的积累是非常好的事儿!

颜色的设计是个问题，不过有好工具来用——Red Alt - I Like Your Colors。图 KDay3-7 演示了 redalt.com 上的 CSS 颜色分析工具。

访问地址: *http://www.redalt.com/Tools/ilyc.php*

精巧地址: *http://bit.ly/91GT9*

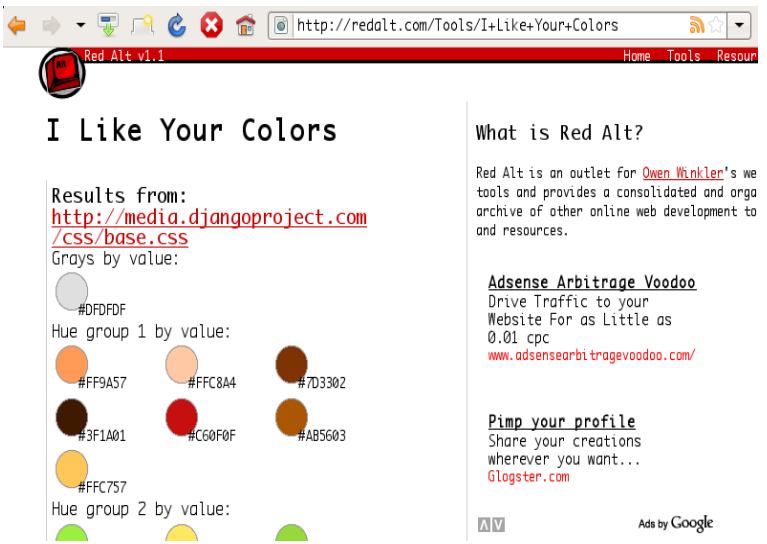


图 KDay3-7　演示 redalt.com 的 CSS 颜色分析工具

CSS 的使用

每一需要 CSS 美化的页面都加<<k_base>> 引用，引用都是从**复用代码**中 clone 过来的同一节点包含的就是 k_base.css 小白根据 Django 社区的样式修剪成的一个通用 CSS 设计，套用到当前的系统中，如图 KDay3-8 所示。



图 KDay3-8　问卷管理和展示 CSS 效果

小结

今天发现并引入了 KQF 和 JVF，好像有点复杂的样子。最终问卷效果如图 KDay3-9 所示。

当然在小白的开发故事中，这不是最终的方案，精彩还在继续……

现在所有的基本功能点都有了，接下来的就是要实用化：

1. 支持多用户，得有用户验证，要登录；
2. 支持多问卷选择/回答/编辑。

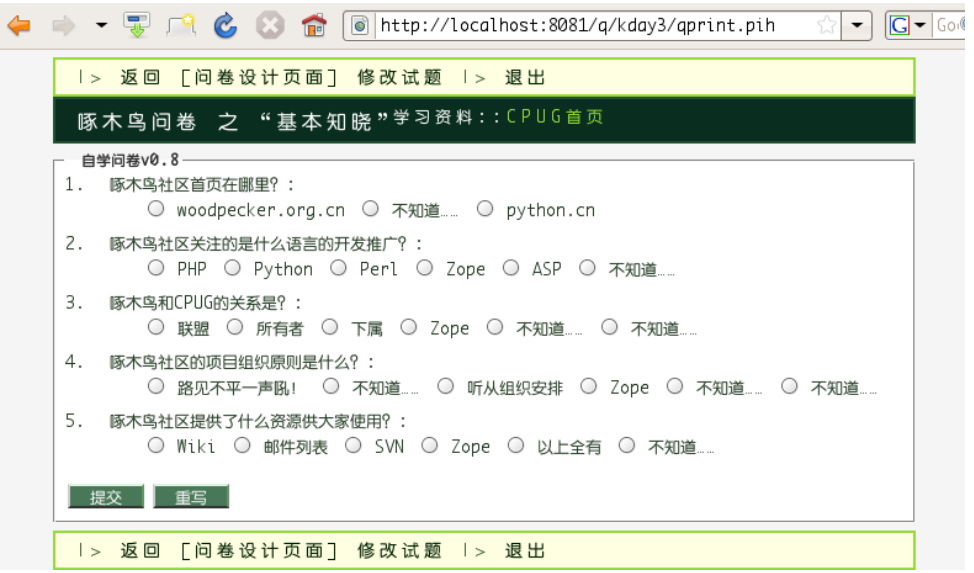


图 KDay3-9　最终问卷效果

实例下载

实例下载的 SVN 下载地址如下所示。

实例下载：*http://openbookproject.googlecode.com/svn/trunk/LovelyPython/KDays/kday3*

精巧地址：*http://bit.ly/2OscFJ*

注意有在上层安置的 JS-JVF 目录，网址为：

http://openbookproject.googlecode.com/svn/trunk/LovelyPython/KDays/js

精巧地址: *http://bit.ly/2lzYbl*

来看一下本日的成果。

```
kday3
|-- Karrigell_QuickForm.py                  KQF 快速表单模块
|-- tryKQF.py                               KQF 试用页面
|-- dict4ini.py                             Limodou 贡献的 ini 解析模块
|-- q/                                      问卷设计文本收藏目录
|   |-- easy051201.cfg                      第一份问卷设计
|   |-- easy051201.cfg.080824231959         问卷设计历史存档
|   `-- easy051201.cfg.080824232025
|-- mana.pih                                问卷设计页面
|-- qdesign.py                              问卷设计实际行为脚本
|-- qpage.pih                               问卷复审页面
```

```
|-- qpage.py                    问卷复审实际行为脚本
|-- qprint.pih                  问卷展示页面
|-- qprint.py                   问卷展示实际行为脚本
`-- questionnaire.tmpl          问卷展示模板
```

练习

在前面章节练习 mysite 的基础上，完成以下功能：

- 利用 Karrigell_QuickForm 改写编辑页面（edit），增加更多的文章属性，如日期，标签，引用链接等；
- 实现了上述之后，使用 JVF，依次校验用户输入的有效性。

提示：可以设计一个文章类，多个文章对象可以利用 pickle 保存；最终应该能够实现新建或修改文章并保存入本地文件的功能。

KDay4　使用 KS 模式重构代码

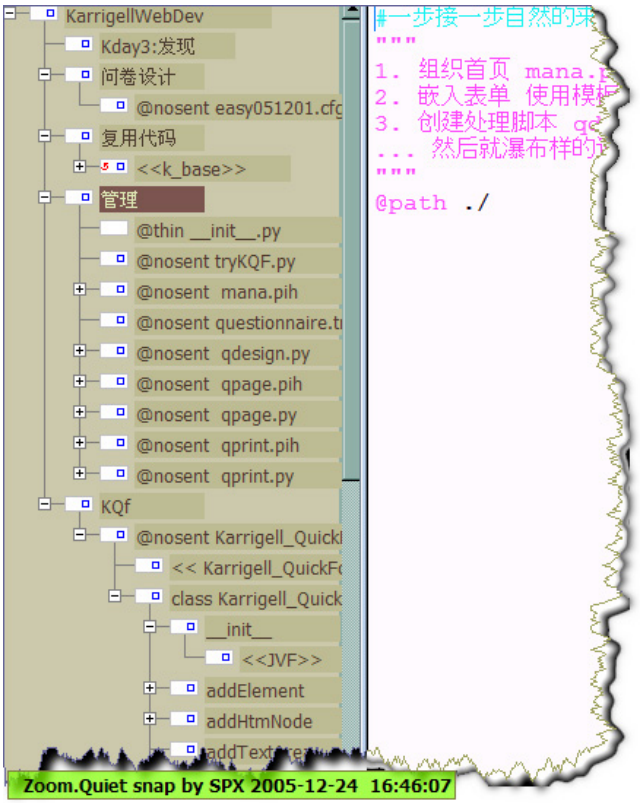


图 KDay4-1　当前开发成果在 Leo 中组织的情景

小白回顾当前的成果（如图 KDay4-1 所示），发现仅实现了三个小功能的原型就动用了七八个文件，这也忒麻烦了，有些想念 CherryPy 的对象式发布了……

可爱的 session

今天，小白计划解决支持多用户和多问卷的问题，先从翻找内置文档开始……

首先要解决的是登录后的用户在不同页面间跳转时，系统知道用户的信息的问题，在小白的记忆中貌似有关 session 的处理可以支持这种需要。

找到相关实例: *http://localhost:8081/demo/ksTest.ks/index*，观察代码，小白追加了自个儿的评注。

```
so = Session()
if not hasattr(so, 'x'):    # 判定 so 对象的 x ~计算属性是否初始化过
    so.x = 0
def index():              # 默认行为,展示所有操作
    print "x = %s" %so.x
    print '<br><a href="increment">Increment</a>'
    print '<br><a href="decrement">Decrement</a>'
    print '<br><a href="reset">Reset</a>'

def increment():          # 累加
    so.x = _private(so.x)
    raise HTTP_REDIRECTION,"index"  # 返回默认展示页面
def decrement():          # 递减
    so.x -= 1
    raise HTTP_REDIRECTION,"index"  # 返回默认展示页面
def reset():              # 置零
    so.x = 0
    raise HTTP_REDIRECTION,"index"  # 返回默认展示页面
def _private(x):          # 不可 URL 访问的支持函式
    """The function name begins with _ : internal function,
    can't be call by a url"""
    return x+1
```

不是吧！！一个页面就作完了四种动作！完全是 CherryPy 味的！！

可以确认 Session 完全是对象了，而且是标准的 KarriGell 内置对象！但是，这要在优雅的 Python 脚本中掺杂 HTML 代码，不爽直！**不过，相比使用 .pih 和包含.py 的方式，使用 .ks 来进行页面组织，已经足够爽直了!**

小白考虑到使用 Cheetah 模板系统，就得每个访问页面都得有对应的模板声明来支持，这也太过于古板和繁琐了，于是小白继续寻求解决方案。

术语 session，一般译作"会话";但是其含义在不同的技术时代和上下文中都是不同的;在我们的实例故事中,特指那种在服务端暂时保存用户状态的 Web 应用技术;这个方面,其他语言一般通过 DB 进行保存和检索;详细情况和其他支持模块的介绍,已经超出了本书的范畴,请读者自行参考相关图书。

HTMLTags

小白猛然发现有函式化的 HTML 生成机制!同样在 KarriGell 内置文档中：*http://localhost:8081/doc/en/htmltags.htm*。

还真的是心想事成！这是 Python 2.2 版本中增加的新功能！可以将一个标准的 HTML 页面由几行 Py 脚本自然生成!

```
1 stylesheet = LINK(rel="Stylesheet",href="doc.css")
2 header = TITLE('Record collection')+stylesheet
3 title = H1('My record collection')
4 rows = Sum ([TR(TD(rec.title,Class="title")
5              +TD(rec.artist,Class="Artist"))
6    for rec in records])
7 table = TABLE(TR(TH('Title')+TH('Artist')) + rows)
8 print HTML(HEAD(header) + BODY(title + table))
```

PCS114 “FP 初体验”中 FP 指的是 Functional Programming 函式化编程，这是和过程化编程不同的思路和体验,笔者在 PCS114 将获得的初步体验进行了初步分享,小白也是在列表中向行者讨教时接触到这一领域的,欢迎勇敢的读者自行品味。

那么，只要结合 KS 和 HTMLTags 功能就可以快速实现想要的复杂展示功能了（如图 KDay4-2 所示）！还可以方便地复用！当然也可使用 Leo 八股化所有的.ks。



图 KDay4-2 KS 模式功能页面在 Leo 中的组织情景

1. `<< page declarations >>` 将引用声明等都归整到一个节点中。
2. `<< htmlcode >>` 就是利用 Leo 来快速复用的各种 HTML 代码。
3. `<<ini>>` 初始化部分就是各响应页面都要使用的变量准备。
4. `@others` 包含以下各种实际运行的功能页面，包含：

- `def action()`：定义的实际响应页面。
- `def_function()`：可以反复调用的不作为页面响应的普通函式声明。

登录、判别

改造思路就是，**从默认的 index 根据用户状态，自动导向 login 页面**。

参考 session 例程中的相关代码：

```
1 ...
2 def increment():          # 累加
3     so.x = _private(so.x)
4     raise HTTP_REDIRECTION,"index"  # 返回默认展示页面
5 ...
```

照猫画虎， 使用 `raise HTTP_REDIRECTION，"login"` 来创立小白自个儿的智能导向登录响应页面。

```
1 def index():
2     << pagehead >>
3
4     if sess.usr["name"]=="NULL":       # 查问 session 对象中的 usr 属性是否
                                           初始化过?
5         raise HTTP_REDIRECTION,"login"
6     else:
7         pass
```

当然，要在 .ks 的`<<ini>>`初始化部分先声明会话容器：

```
1 # 使用 Session 来记忆成员信息
2 sess = Session()
3 if not hasattr(sess, 'usr'):
4     sess.usr = {"name":"NULL"}
```

在没有进行登录前，准备一个值为 `NULL` 的 `usr` 用户会话对象。

好了，实际的登录表单还是使用 KQF 来实现：

```
 1 p = Karrigell_QuickForm('fm_login'
 2                     ,'POST'
 3                     ,'chkusr'
 4                     ,"登录自学问卷")
 5 p.addHtmNode('text','uname'
 6            ,"staff 帐号"
 7            ,{'size':40,'maxlength':8})
 8 p.addGroup(["submit","btn_submit","提交","btn"]
```

```
 9          ,["reset","btn_reset","重写","btn"])
10 p.addRule('uname','required'
11          ,"成员名是必须的!Login name is required!")
12 p.display()
```

嗯，这比使用模板要直观、简洁、整齐。然后进行有效性判别的处理，这也是个标准的 KS 函数。

```
 1 def chkusr(**args):
 2     """检查用户登录情况
 3     """
 4     print QUERY
 5     sess.usr["name"] = QUERY["uname"]
 6     if sess.usr["name"] in pmguys:
 7         # pmguys 是个元组,预定义的有管理整理的成员帐号
 8         sess.usr["pm"] = 1
 9     print sess.usr
10     #raise HTTP_REDIRECTION,"index"
```

先别让页面自动跳走，看一看，是否如愿地记录了登录信息？（如图 KDay4-3 所示）

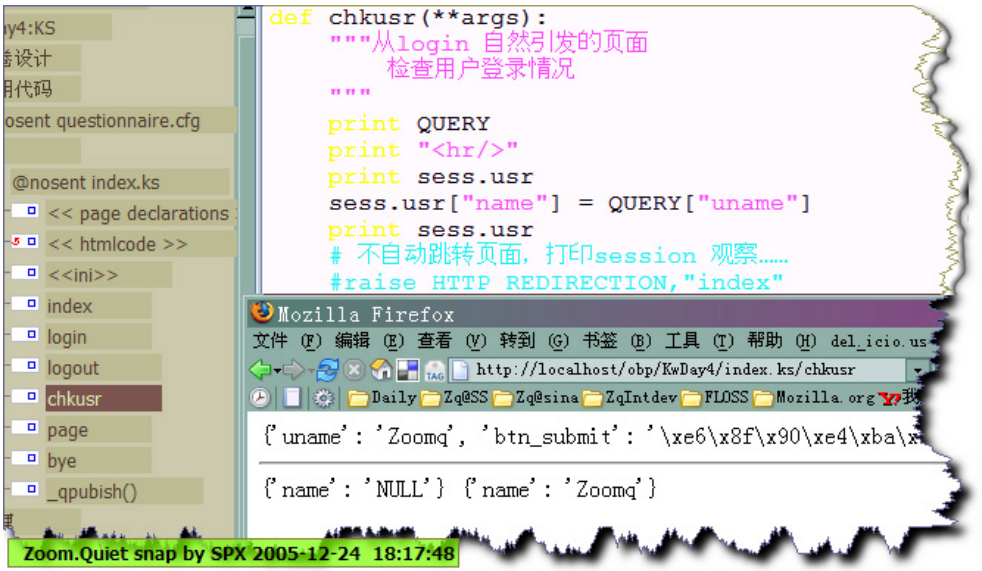


图 KDay4-3　根据 session 值自动判别用户身份

这里我们再回顾一下问卷的特点。问卷的使用原则是，任何人都可以登录/回答，但是：

1. 有效的成员问答才会统计；
2. 只准个别的成员拥有管理/编辑问卷的权限。

因此，我们必须考虑多问卷列表选择的问题，目前有了登录后的用户信息，就可以根据不同的角色进行处理了。

fnmatch 模块

小白首先要解决的问题就是，在指定目录中寻找依照某约定模式命名的所有文件。

这次不用询问行者就知道，必定有内置的支持模块！果然有——fnmatch，配合设置进行：

```
1 qcfglist = []
2 for f in os.listdir(conf.qpage.qpath):
3    if fnmatch.fnmatch(f, '*.cfg'):
4       if ("__init__" in f):
5          pass
6       else:
7          qcfglist.append(f)
8 print qcfglist
```

使用以上代码就可以搜索出目录中所有以.cfg 结尾的文件!conf 对象是新发布模式下的统一设置信息对象。

因为担心可能需要多个.ks 文件来组织网站，不愿意像原先那样滥用 Leo 来复用信息了，所以，建立@nosent questionnaire.cfg 统一配置文件。这样，通过 conf = DictIni ("question naire.cfg")，就可以在所有.ks 页面中获取统一的站点设置了。

列表显示所有问卷

根据问卷的状态进行归类列表。

```
 1 # 识别问卷发布情况::
 2 qdone = {}
 3 qdoing = {}
 4 qdesign = {}
 5 for p in qcfglist:
 6    cfgp = DictIni(conf.qpage.qpath+p)
 7    if 0==cfgp.desc.done:
 8       qdesign[p]=cfgp
 9    elif 1==cfgp.desc.done:
10       qdoing[p]=cfgp
11    elif 2==cfgp.desc.done:
12       qdone[p]=cfgp
```

```
    else:
        qdesign[p]=cfgp
```

使用不同的字典容器先过滤搜索出来的文件；然后使用类似的输出。

```
print H4("问卷进行中::")
print UL("".join([str(LI(
          B(A(qdoing[i].desc.pname
              ,href="page?qpname=%s&do=doing"%i.split(".")[-2])
              )+
              SUP(qdoing[i].desc.learn)+
              SUB(A("问答统计",href="stat?qp=%s"%i))
                    )
                  ) for i in qdoing.keys()
                ])
                )
```

得到的结果如图 KDay4-4 所示。



图 KDay4-4　多问卷列表管理时情景

小结

今天，小白使用 .ks 体验了一把**函式化页面发布**，成果就一个脚本 index.ks ，当然，以前完成的.pih 功能页面依然好用，而小白将利用今天的成果将它们都重构为更加爽快的**函式化页面发布**，而且，小白忍不住对 CSS 进行了深入的定制，在输出代码的最简化和页面表现丰富之间进行了有效的探索。

又可以看看本日成果了。

```
kday4/
|-- Karrigell_QuickForm.py              KQF 快速表单模块
|-- dict4ini.py                         Limodou 贡献的 ini 解析模块
|-- q/                                  问卷设计文本收藏目录
|   |-- CPUG051211.cfg                  多问卷实例-CPyUG 社区问卷
|   |-- Python051221.cfg                -Python 基础问卷
|   `-- easy051201.cfg      -           实验问卷草稿
|-- index.ks                            使用 KS 模式重构的有登录功能的系统首页
|-- mana.pih                            问卷设计页面
|-- qdesign.py                          问卷设计实际行为脚本
|-- qpage.pih                           问卷复审页面
|-- qpage.py                            问卷复审实际行为脚本
|-- qprint.pih                          问卷展示页面
|-- qprint.py                           问卷展示实际行为脚本
|-- questionnaire.cfg                   问卷系统配置文件
|-- questionnaire.tmpl                  问卷展示模板
`-- tryKQF.py                           KQF 试用页面
```

明日任务

明天的任务是什么呢？快速地完成更加简练的功能页面！但是，如何将编辑、回答、统计融合起来？

1. 点击进入不同问卷，可以回答/或是编辑；
2. 点击编辑就直接编辑；
3. 提交就成为新版本问卷；
4. 问卷要有版本管理；
5. 提交后，可以立即回到问答或是问卷集首页；
6. 任何页面都有安全退出的链接。

唉呀，好象任务越做越多了……

实例下载

实例下载请使用 SVN 下载的地址。

下载地址：*http://openbookproject.googlecode.com/svn/trunk/LovelyPython/KDays/kday4*
精巧地址：*http://bit.ly/1OMMQt*

练习

在前面章节练习 mysite 的基础上，实现以下功能：

- 我们在学习了.ks 之后，将原来的 mysite 改写成函式化页面发布，这样我们只需一个 index.ks 文件就可以完成之前 index.pih 和 edit.pih 两个页面的功能；
- 增加用户登录、 验证和登出功能，即实现用户通过输入姓名和密码，提交并通过验证后登录首页，具体可参考 png 中截图；
- 使用 HTMLTags 美化首页显示，读者可以展开自己丰富的想象力啦。

KDay5　通过 session 重构应用流程

快速利用已有代码，运用体会到的所有技巧，快速重构功能实现！

昨日小白通过测试 .ks 模式的发布方式，确认这样可以减少代码文件数量，而且通过 Leo 可以快速重构各种功能，也可以让原先分散的.pih 们也整合到一个 mana.ks 事务页面中!

悠然 Leo

Clone 节点功能，在 Leo 中的快捷键是 Ctrl+`，其操作界面如图 KDay5-1 所示。

从来没有什么编辑器的操作令小白如此自信过！因为只要保证<<htmlcode>> <<pagehead>> <<pagefoot>>这三个节点是 Clone 来的，那么，就等于统一了所有页面的外观! <<htmlcode>>节点收集了 HTML 相关的标准头/尾输出字串，以及 CSS 样式定义；<<pagehead>>和<<pagefoot>>节点是分别利用<<htmlcode>>节点中的相应定义输出定制的页面头/尾。而且这三个节点，可以单独收纳在另外的节点中，专门进行维护!

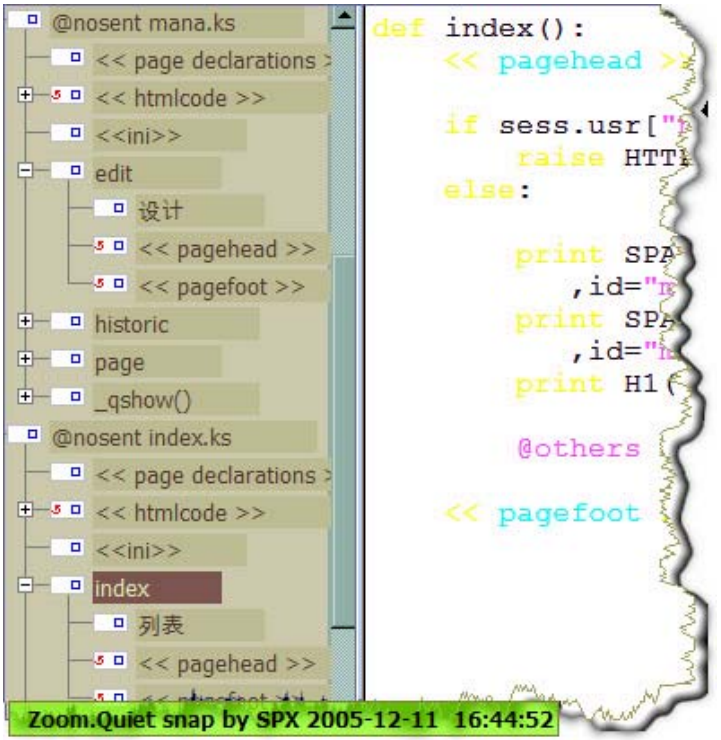


图 KDay5-1　在 Leo 中运用克隆节点的情景

调整事务响应

现在，小白决定将问卷的使用和管理拆分为简单明确的两个事务页面：

- `index.ks` 文件对应，默认访问页面，是问卷回答/统计的使用事务页面；
- `mana.ks` 文件对应，有权限用户的问卷设计管理事务页面。

那么，首先小白得解决：用户从`/index.ks/login` 登录后的个人信息，怎样传递给`mana.ks`？

页面间传递对象

当然，小白的期望是 Pythonic 式地传递完整对象，而不用判别/编码/DB 中转等。行者指点过：“不是有 `pickle` 序列化模块嘛！”小白猜测，将对象序列化为字符串，作为页面间的 URL 参数捅过去就好啦!

打印当前的会话容器序列化为字串看一看，如图 KDay5-2 所示：

```
print pickle.dumps(sess.usr)
```

PCS210

PCS210“pickle”支持将对象导出成硬盘文件以便进行网络传输等其他进一步操作;在 PCS210 中,分享了进一步操作技巧



图 KDay5-2　将原生序列化对象打印到页面的情景

唉呀！有空格以及其他符号！这些内容一定会被解析为非期待参数的，那就得使用 URL 安全编码了，找呀找呀，找朋友，找到一个 `urlsafe_b64encode(s)`是 base64 模块中的。OK！组织测试一下（如图 KDay5-3 所示）：

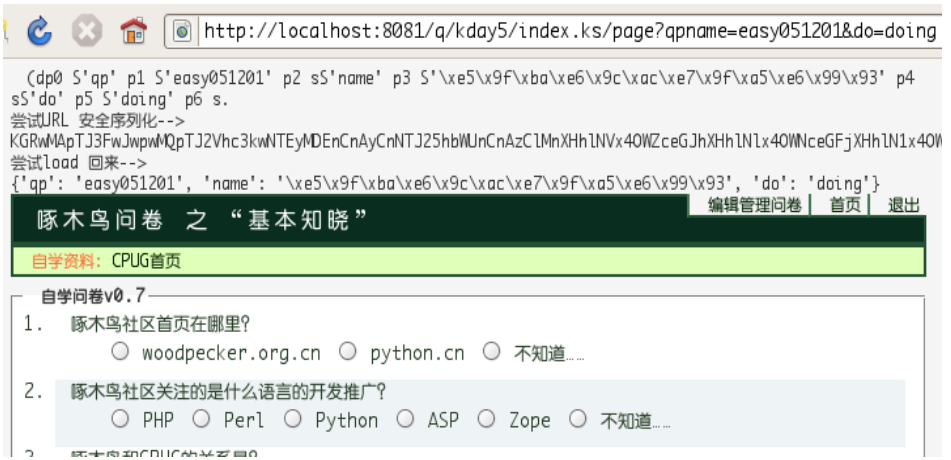


图 KDay5-3　原始和编码后的对象序列打印到页面的对比情景

```
1 # 序列化对象以便页面间传递
2 pisess = pickle.dumps(sess.usr)
3 print pisess
```

```
print "<br/>尝试 URL 安全序列化--> "
sessurl = base64.urlsafe_b64encode(pisess)
print sessurl
print "尝试 load 回来--> <br/> "
print pickle.loads(base64.urlsafe_b64decode(sessurl))
```

非常理想！可以将所有用户状态信息包含在一个字典对象中，而字典对象可以编码成 URL 安全的格式，通过链接在任何页面中被 Karrigell 重新解码并重新载入成标准的字典对象，就可以进行任意操作了。

先将设计中的富信息链接输出到页面的~菜单生成容器中：

```
print SPAN(A('> 编辑管理问卷'
       ,href="../mana.ks/edit?qp=%s&obj=%s"%(
          qpname
          ,base64.urlsafe_b64encode(
             pickle.dumps(sess.usr))
             )
          )
       ,id="mana")
```

确认无误后，将 标签清除，就等于在问卷回答界面中追加了一个编辑的入口链接，效果如图 KDay5-4 所示。



图 KDay5-4　问卷列表和含有页眉控制菜单的问卷页面

注意：

1. 使用 `../mana.ks` 是因为 .ks 事务页面实际都是进一层的`*.ks/action` 访问，所以要链接到另外的事务页面,要使用相对路径修正。

2. 对<A>追加了一个 id="mana" 的属性是为了让 CSS 集中，进行显示，对应的 CSS 定义是：

```
SPAN#mana{border-bottom: 2px solid #778;
   border-left: 2px solid #778;
   float:right;color: #930;
   background-color: whitesmoke; padding: 2px;}
```

当然，应该只有权限用户才会看到此链接。那么只要简单地在菜单生成部分追加个用户身份的判定就可解决这个问题；只是怎么管理和组织用户权限组呢?这个明天再说，不过，在开发阶段完全开放——反正将实现问卷的版本管理，这会儿不用怕的。

至此，小白通过访问链接的参数，携带了足够的用户信息，来支持各种判定和外部文件的读取；完成了针对不同用户进行不同响应的关键 Web 应用功能； 然而所需要的仅仅是一个内置模块的支持，以便随手打印到页面的脚本调试方式，页面调试方面，就 Python 的 Web 应用调试而言，非常直接! 因为 Python 的内置自省能力，可以在运行时的任何一刻使用 dir() 函式，将当前内存空间中有关对象全部打印出，以便进行观察； 小白一直是使用这种土办法进行调试的，唯一发现的技巧就是：

- 必要时使用 <PRE/>～HTML 引用标签将输出字串包起来，以便原样显示，以免有时候 Python 的输出含有<>字符前后包围时，被游览器误解为 HTML 标签而试图解析，结果在页面上看不出，得使用“查看源代码”方式，从 HTML 原始输出中才能观察到!

重构编辑操作

所谓 mana.ks 事务页面就是要完全替代之前使用 7 个文件才能完成的问卷编辑流程。小白二话不说先将成功运行的 index.ks 节点在 Leo 中用 Ctrl+Shift+C 和 Ctrl+Shift+V 复制成 mana.ks 节点，然后快速地修剪，再将原先 qpage.py 的 expage(dict) 重构为 index.ks 的 _qshow(dict，aim) ，99% 的代码都不变，仅仅对传入的参数重新命名而已。

```
1 def _qshow(dict,aim):
2    """将 dict 内容输出为回答问卷
3    """
4    exp = ""
5    p = Karrigell_QuickForm('fm_kq','POST',aim,dict.desc.desc)
6    p.addElement('node','<ul>','')
7    # 深入数据
8    qli = {}
9    k = [int(i) for i in dict.ask.keys()]
```

```
    k.sort()
    for i in k:
        ask = dict.ask[str(i)]
        qk = [j for j in ask.keys()]
        qk.sort()
        for q in qk:
            if 1==len(q):
                qli[q] = ask[q]
            else:
                pass
        question = ask["question"]+"<sup>答案::%s</sup>"%ask["key"]
        ## 这里美化一下子,奇偶行使用不同的 CSS 来控制
        if i%2 == 0:
            p.addRadioList("cr_ask%s"%i
                  ,question
                  ,qli
                  ,"even")
        else:
            p.addRadioList("cr_ask%s"%i
                  ,question
                  ,qli)
        p.addJSRule("cr_ask%s"%i,"问题%s "%i)

    p.addJSValidation()
    p.saveJSRule("js/validation-config.xml")
    p.addElement('node','</ul>','')
    p.addGroup(["submit","btn_submit","提交","btn"]
             ,["reset","btn_reset","重写","btn"])
    exp += p.export()
    return exp
```

PCS212 “ shutil ”介绍了这一针对文件操作的高级支持模块的基本使用技巧;对 copy2() 这一函式好奇的读者可以进入 PCS212 进一步探寻。

同理，使用原先十分之一的时间，快速将一连串的 .pih 动态页面改写为 .ks 事务页面，只是为了增加版本管理的功能，然后追加一个操作，先复制 `shutil.copy2(qpage，qpage+".%s"%time.strftime("%y%m%d%H%M%S"， time.localtime()))`。

使用 shutil 高级文件处理模块，在问卷提交后，先复制一份历史版本。

历史版本

小白是领教过啄木鸟社区的维基的，习惯了维基式版本管理的反悔模式，当然也想自个儿弄一个，以便在问卷编辑过程中可以回退到任意一个编辑版本中。图 KDay5-5 则显示

了问卷历史版本访问列表的原始情景。

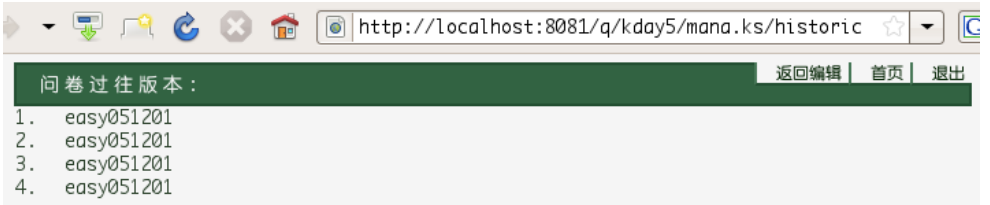


图 KDay5-5　问卷历史版本访问列表的原始情景

经过几次小的尝试性修改，在 /q 中聚集了一批命名有模式可寻的问卷设计稿，文件就是前述进行历史版本自动记录时的后缀设定：

```
".%s"%time.strftime("%y%m%d%H%M%S", time.localtime())
 | |   |     |       |                      +-- 生成当前时间对象
 | |   |     |       +-- 时间规格字串,这儿的含义是 年,月,日,小时,分,秒
 | |   |     |             紧跟着拼一起
 | |   |     +-- 时间格式化输出,将当前时间按规格输出
 | |   +-- 使用 time 模块中的支持函式
 | +-- 字段模板替换算子和引号后%引领的对应配合,将对象的内容以字符串的格式输出到对应位置
 +-- 模板字串,这里要求输出 .080916225229 格式的文本字串
```

PCS213

PCS213“ time ”进一步分享了这一对时间相关操作支持的非常到位的常用模块的具体使用技巧。

直接利用问卷列表的处理代码，小修一下，就成为历史版本列表的实现，归到：

```
1 def historic(**var):
2     << pagehead >>
3     @others
4     << pagefoot >>
```

的 @others 节点中，小白注意到，自个儿已经很自然地将事务页面中每个具体请求响应部分的代码也八股化了。

1. <<pagehead>> 节点包含标准的页头信息。
 不过是一个 print _htmhead("**标题文字**") 将预定 HTML 代码少量处理后输出的小函式。
2. @others 包含所有具体行为的章节脚本。
3. <<pagefoot>> 节点包含标准的页底 信息。
 更加简单的 print htmfoot 打印行为，将 Powered by:...什么的广告全都统一而已。

关键代码

具体的历史版本问卷列表输出代码如下：

```
qnow = fnmatch.filter(os.listdir(conf.qpage.qpath)
                      , '%s.cfg.*'%sess.usr["qp"])
# 从相关目录搜索出相关文件
print "<UL>"
qnow.reverse()
for l in qnow:
    s = l.split(".")
    print LI(A("""%s
            <sup>%s/%s/%s %s:%s:%s 版本</sup>
            """%(s[0]
            ,s[-1][:2]
            ,s[-1][2:4]
            ,s[-1][4:6]
            ,s[-1][6:8]
            ,s[-1][8:10]
            ,s[-1][10:]
            ),href="edit?his=%s"%s[-1]
               )
            )
# 分解文件信息,组织成访问链接
```

老技巧，使用 `fnmatch` 模块将吻合模式的文件搜索出来，使之成为一个列表对象。

现在的模式是：`'%s.cfg.*'%sess.usr["qp"]`

即，想要看的问卷名+".cfg"+.**历史标识字串**. **历史标识字串**就是之前随手复制时，设定的`time.strftime("%y%m%d%H%M%S", time.localtime())`时间字串，小白自个儿设定的是**年月日时分秒**的组合，使用字串的切片操作，一下子就重新组织好了输出（如图 KDay5-6 所示）。

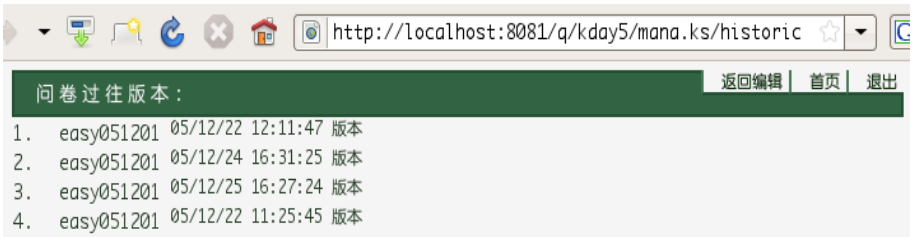


图 KDay5-6　历史版本时间点附加输出成功情景

再创建一个链接，追加一个?his= 的历史版本参数标识，立即回头修改一下。

```
def edit(**var):
    #...
    if "his" in QUERY.keys():
        #进入历史版本编辑
        qpage = "%s%s.cfg%s"%(conf.qpage.qpath
                  ,sess.usr["qp"]
                  ,"."+QUERY["his"])
    else:
        qpage = "%s%s.cfg%s"%(conf.qpage.qpath
                  ,sess.usr["qp"]
                  ,"")
    #...
```

在编辑行为中，追加个小判定，令编辑处理知道什么时候读指定的那个历史版本文件。

可变参数的意义

现在小白可是对 Python 函式的可变参数深以为然了，本来按照 PHP 的习惯，传递的页面参数自然会变成内存对象给脚本来引用。当然，行者评点说，这是以必须有 Apache 等 Web 服务器的配合，直接在内存中折腾作代价换来的，但是在 KarriGell 中如果不严格匹配就会有如图 KDay5-7 所示的参数不匹配的错误情景。

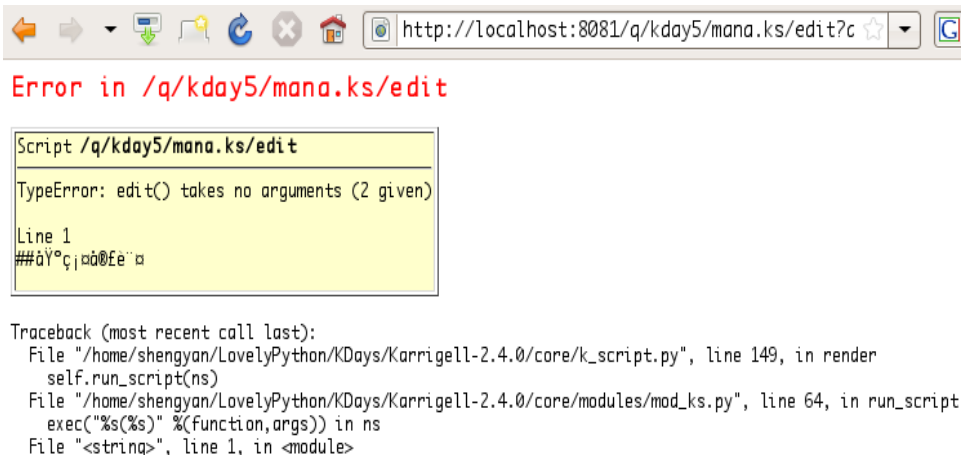


图 KDay5-7　参数不匹配的错误情景

PCS108“ 函式 ”分享了函式这一 Python 程序的基本单位的组织体验和技巧;
PCS109“ 系统参数 ”则关注函式参数的使用技巧。

但是有了 Python 中万能的(**var) 声明，一切都总能兼容!

进一步的改进

小白认为应该可以列出各版本的内容体积情况，继续到 Python 模块手册中游荡一下，发现了 os.stat()，先追加到代码中。

```
    <sub>大小:%sb</sub>
    ...
    ,os.stat(conf.qpage.qpath+l)
```

输出如图 KDay5-8 所示。

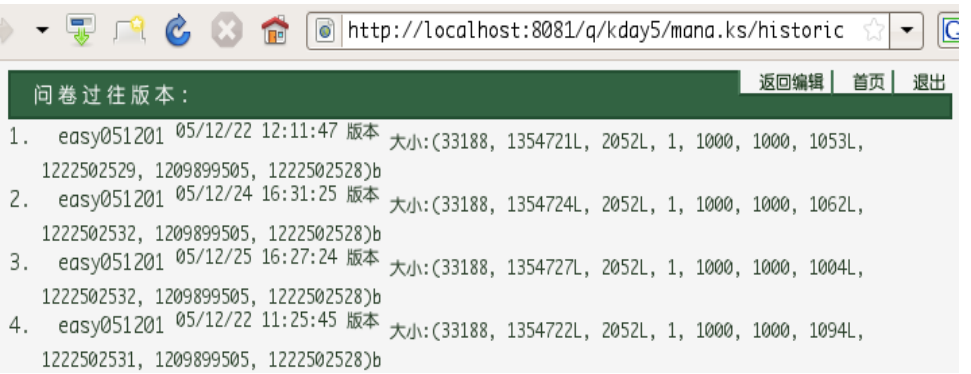


图 KDay5-8　原始 os.stat()输出字串

返回元组有文件对象的如：

```
    st_mode (protection bits),
    st_ino (inode number),
    st_dev (device),
    st_nlink (number of hard links),
    st_uid (user ID of owner),
    st_gid (group ID of owner),
    st_size (size of file, in bytes),
    st_atime (time of most recent access),
    st_mtime (time of most recent content modification),
    st_ctime (platform dependent; time of most recent metadata change on Unix,
    or the time of creation on Windows).
```

ST_SIZE 是小白想要的，但是具体是哪个？看标识，只有两个是 L 长整型，有变化的就倒数第 4 个，所以：

```
    <sub>大小:%sb</sub>
    ...
```

```
    ,os.stat(conf.qpage.qpath+l)[-4]
```

得到图 KDay5-9：



图 KDay5-9　os.stat()输出结果合理格式化后情景

小白真感受到了心想事成般的自在了。

PCS200 “os(.stat;.path)”介绍了相关系统信息获取支持的常用模块的进一步操作技巧。

小结

小白在昨日基础上快速使用 .ks 模式服务页面，将原先实现了的功能全部归并到了两个文件中，进一步地：

1. 成功追加了不同问卷的编辑入口，通过链接参数，令编辑页面知道读入哪个文件。
2. 成功增强了问题编辑的版本管理，通过支持模块快速复制原有文件，并组织好历史版本访问列表。

完全达到了昨天的计划开发目的!

同样来看看本日成果。

```
kday5/
|-- Karrigell_QuickForm.py            KQF 快速表单模块
|-- dict4ini.py                       Limodou 贡献的 ini 解析模块
|-- q/                                问卷设计文本收藏目录
|   |-- CPUG051211.cfg                多问卷实例-CPyUG 社区问卷
|   |-- Python051221.cfg              -Python 基础问卷
|   |-- easy051201.cfg                -实验问卷草稿
|   |-- easy051201.cfg.051222112545   -实验问题编辑历史快照
|   |-- easy051201.cfg.051222121147
|   |-- easy051201.cfg.051224163125
|   -- easy051201.cfg.051225162724
|-- index.ks                          使用 KS 模式重构的有登录功能的系统首页
|-- mana.ks                           使用 KS 模式重构问卷设计综合支持页面
|-- mana.pih                          问卷设计页面
```

```
|-- qdesign.py                    问卷设计实际行为脚本
|-- qpage.pih                     问卷复审页面
|-- qpage.py                      问卷复审实际行为脚本
|-- qprint.pih                    问卷展示页面
|-- qprint.py                     问卷展示实际行为脚本
|-- questionnaire.cfg             问卷系统配置文件
|-- questionnaire.tmpl            问卷展示模板
`-- tryKQF.py                     KQF 试用页面
```

明日任务

唉呀！没有想到无论多古怪的想法 Python+KarriGell 都支持！接下来呢？就是要根据有效的成员信息来进行成绩的统计汇报了！

实例下载

实例下载请使用 SVN 下载地址。

实例下载：*http://openbookproject.googlecode.com/svn/trunk/LovelyPython/KDays/kday5*

精巧地址：*http://bit.ly/1uZlq3*

练习

在前面章节练习 mysite 的基础上， 完成以下功能：

- 在首页增加删除链接，实现删除指定文章的功能；
- 增加搜索功能，即输入关键字，进行查找后得到各个文章对象并显示到结果页面中；
- 能否优化我们现在的整个网站框架，若能够则进行最大化的结构优化， 读者可以根据已经学到的知识自由发挥。

KDay6　利用 mm 人性化组织成员信息

此 mm 非彼 MM，它乃指 FreeMind 思维图谱文件，因默认后缀是.mm 而得名。

FreeMind 的另类用法

继承原先问卷系统的要求：问卷成绩的统计是针对团队的。所以，得有方法知道回答者的成员组织信息，以便对应分组统计。

小白第一时间想到了 FreeMind，是因为，如果使用 DB，怎么都得立成员表和组织表，然后还得根据 ID 外键关系等进行对应选择，好麻烦的，还是经过行者推荐、轻松使用起来的 FreeMind 直观，操作也非常快捷——复制，移动太爽直了！关键是，FreeMind 文件本身其实就是 XML 格式的文本！本质上这也是种数据库，可以直接使用。而且行者们曾经提供过 `freemind.xsl+freemind.mm` 可以配合在浏览器中输出 mm 文件（如图 KDay6-1 所示）。

XSL 模板其实无法直接解析.mm 文件，得手工在头部追加个 xml 头声明，类似：
```
<?xml version="1.0" encoding ="UTF-8"?>
<?xml-stylesheet type='text/xsl' href='freemind.xsl'?>
```
以便通知浏览器套用什么 XSLT 模板。

哈哈哈！由此，小白大致知道了 FreeMind 的数据格式：

1. 所有节点是统一的 `<node>`。
2. 属性和值，全部是 UTF8 编码的字串。

所以，只要有简单的约定，就可以作为 Xpath 的过滤参数来理解真正的数据意义了，以上的组织成员信息就简单依从以下的约定（如图 KDay6-2 所示）。

一级节点都是部门描述，中文；二级节点是属性描述，如：

- dept 说明部门的整体信息；
- staff 汇集成员信息，类似部门的节点组；
- total 成员总。

PCS402 “XML”介绍了这一风生水起的技术领域，分享了在 Python 中对 XML 处理思路。

PCS403 “思维导图”介绍了导图这一种极高效的思维记录和辅助工具；分享了流行的导图工具，其中包含了对 FreeMind 这一轻盈完备的导图工具的愉快体验!

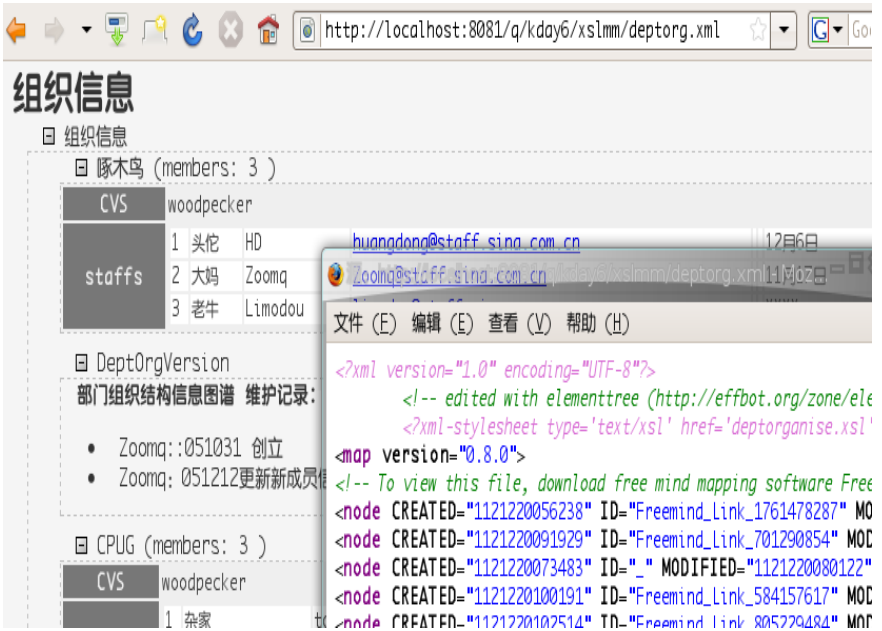


图 KDay6-1 .xsl 解析手工增补后的.mm 文件输出时的情景

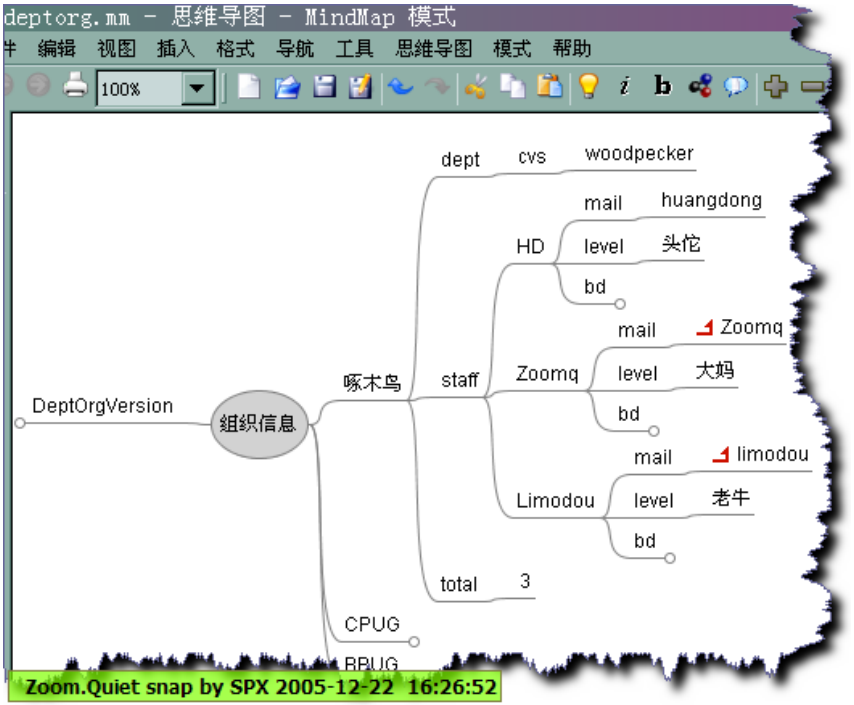


图 KDay6-2 FreeMind 中根据约定组织好的团队信息

这样就令 XSLT 有基础可以进行数据组织了，同理，也可以让 Python 快速理解了。

注意：XSLT 是以 XML 为形式用来解析 XML 数据的样式语言，其实它是 XML 应用领域中很实用的一个标准；Xpath 就是 XML 中的查询语句，可以快速从 XML 数据岛中过滤出想要的信息。在 PCS402 XML 中有相关的介绍，读者可以深入了解。

当然 ElementTree

在 Python 世界，处理 XML 解析的模块有多种，小白从行者们那儿知道其中 ElementTree 最好用！

不仅仅因为 ElementTree 好用到已经被收为官方标准包，而且支持 Xpath 的搜索，其实从使用方面也是 ElementTree 最 PythyonIc 了，`deptorg.py` 是根据对 .mm 的理解，快速组织的专用解析脚本，其中：

```
1 from elementtree import ElementTree
2 xmlFileName = "deptorg.xml"
3 print open(xmlFileName,"r").read()
4 tree = ElementTree.parse(xmlFileName)
5 elem = tree.getroot()
6 dept = elem.findall("node/node")
7 for d in dept:
8     print d
9     print LI(d.attrib["TEXT"].encode("utf8"))
```

代码注解：

1. 引入 ElementTree 模块。
2. 读入指定 XML 文件。
3. 处理 XML 文件为 ElementTree 对象。
4. 找到 XML 根。
5. 从根找到所有第二级的<node>节点对象列表。
6. 找到的所有节点，组织成一个列表，已经可以循环处理并通过 `.attrib["TEXT"]` 来获得内容了！

PCS402 “XML”中有相关的介绍，读者可以深入了解。

注意：

"node/node" 就是一句 Xpath 语句，要求 XML 解析器，找出所有<node>标签中次一级的<node>标签节点!

图 KDay6-3 显示了页面输出和 XML 文本对照的情景。

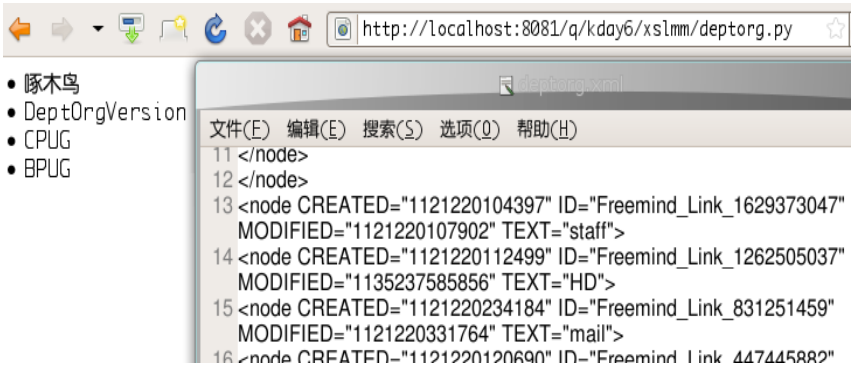


图 KDay6-3　页面输出和 XML 文本对照情景

使用 `findall()` 和 `attrib["TEXT"]` 便可以从 .mm 中判别和获取足够的信息，并输出数据到页面了!具体的请读者去看相应脚本吧。

PCS214“ElementTree“进一步介绍了这一因为自身的优秀品质，被吸收为内置模块的第三方 XML 处理模块，分享了相关的常用操作体验。

通过 SVN 地址下载：

http://openbookproject.googlecode.com/svn/trunk/LovelyPython/KDays/kday6/xslmm/deptorg.py

精巧地址:*http://bit.ly/15OMam*

在 Karrigell 中混合对象

怎样可以在问卷系统中使用 .mm 的理解成果?

麻烦在于 `ElementTree.parse()` 需要 XML 文件的绝对访问路径，然而，小白不想在程序中包含太多系统的真实路径信息，所以，反复尝试，发现从相对路径偏移还是从 URL 来访问都不靠谱，最简单的办法就是将解析脚本和 .mm 文件加在一起。但是问题就来了，各种事务页面如何快捷直观地使用理解成果? 自然会想到 `Include()`KarriGell 中到处都可以进行的基础操作，包含：

```
Include("../xslmm/deptorg.py")
                ^
                |
                +- 使用相对路径,回退一级;
                从 index.ks/login 之类的下级行动函式页面访问
```

小白根据 MoinMoin 包含操作的经验，猜想着这样一来应该能将字典对象混入当前名称空间吧？于是在专门的 .mm 解析脚本——`deptorg.py` 中尝试 `print dir()`。图 KDay6-4 显示了在页面输出当前名称空间的情景。

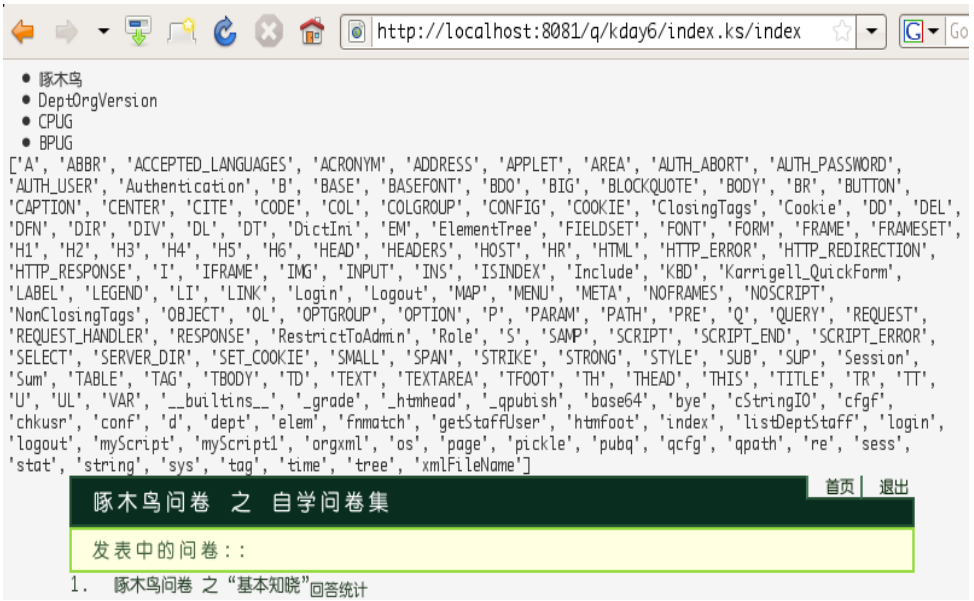


图 KDay6-4　在页面输出当前名称空间的情景

不会这么简单吧！小白发现了本来应该只在 `index.ks` 事务页面中才有的对象 sess ——对话容器，继续尝试赋值，在 `deptorg.py` 中最后追加：

```
.....
sess.usr["dept"]=deptall
sess.usr["deptree"]=deptree
```

真的好用！在 `index.ks/stat` 统计页面，包含 `deptorg.py` .mm 专用解析脚本，再打印 sess.usr 看一下，真的有了！如图 KDay6-5 所示。

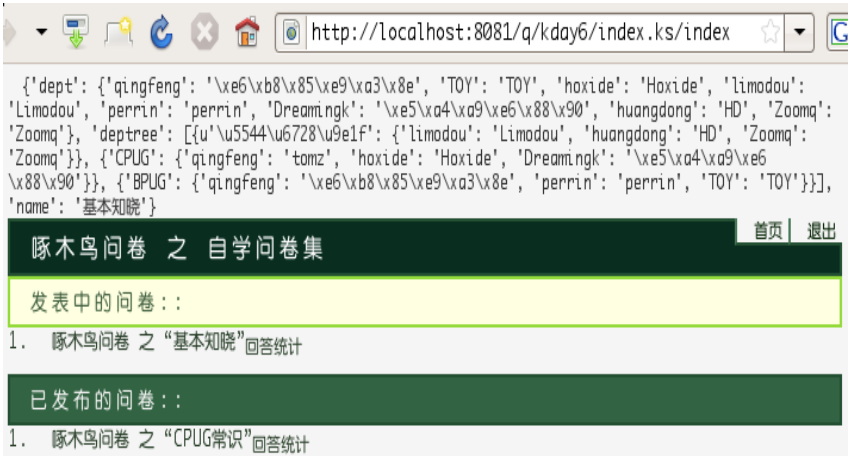


图 KDay6-5　在页面输出从 .mm 文件中解析出来的成员信息

想要的数据从.mm 解析出来了，传递给了页面，也测试打印出了。

表单验证

进行回答统计前，必须有个先决重构。

有效的答案必须是全部问题都回答了，不然的话不仅会产生用户反复问答的情景，而且统计会不准确，但是曾经令小白激动的 JVF 忽然间怎么也不好用了！不得已，仔细看了一下子代码，才发现有好几处设计是不能容许的。

- 所有表单都要读取同一配置 XML，无论访问者想回答哪个问卷，都会导致动态生成配置文件时有争用问题。
- 同样的原因，导致每次读取配置文件，可能要读入相对无用的几倍信息，别的表单的检验策略也被迫读取了。
- 在 validation-framework.js 中代码如下：

```
try {
var prefix = ["MSXML2", "MSXML", "Microsoft", "MSXML3"];
for (var i = 0; i < prefix.length; i++) {
       //return new ActiveXObject(prefix[i] + ".DomDocument");
       var obj = new ActiveXObject(prefix[i] + ".DomDocument");
       if (obj == null || typeof(obj) == 'undefined') {
              continue;
       } else {
              return obj;
       }
```

```
    }
    } catch (e) {
    //^_^
    throw new Error("My God, What version of IE are you using? IE5&+ is
    requiered.");
    }
```

啊？BS！居然不能支持 FireFox 的，难道以前小白的成功是偶然现象？！

小白彻底认输！看来只有直接使用服务端的判定，创立 `bye()` 判别处理页面，并在失败时输出：

```
    <input type="button"
        value="点击返回重新回答"
        class="btn"
        onClick="history.back();"/>
```

以 JS 行动按钮，快速完成**必须回答完全部问题之检验**。

如图 KDay6-6 所示，算是基本可用了，只是这样一来，加重了服务端的压力，也无法进行客户端页面的就地提醒，嗯，自古事难万全，先这么着，以后再考虑这个问题吧！



图 KDay6-6　查觉有未回答问题时提醒的情景

统计汇报

终于进入最后的阶段!小白已经开始志得意满起来，因为，下来似乎已没有什么难度，接下来的功能就单纯了：

- 将所有人的回答输出记录成简单的文件，比如说：每个题目的回答为一行；
- 套用问卷列表的技巧，需要时可以搜索出对应问卷的所有成员回答记录，并批量读到列表中，和成员信息字典匹配，就可以人性化地输出了。

相关代码为：

```
 1 ali = fnmatch.filter(os.listdir(conf.qpage.apath), '%s.*.aq'%qpname)
 2 aed = []
 3 for f in ali:
 4     a = open(conf.qpage.apath+f,"r").read()
 5     aed.append(f.split(".")[-2])
 6 done = []
 7 unknow = []
 8 for a in aed:
 9     if a in sess.usr["dept"].keys():
10         done.append(a)
11     else:
12         unknow.append(a)
```

代码注解：

预先处理，过滤出不知道的非期待的回答。

- 行 1: 从目录中查找出所有相关回答记录文件（小白约定回答文件的命名格式是**问卷名、成员账号名、aq**）。
- 行 2~5: 根据文件名快速整理出名单列表 `aed`。
- 行 6~12: 顺势，就有了已知合理成员回答列表 `done` 和未知成员列表。

成员回答情况的汇报总表如图 KDay6-7 所示。

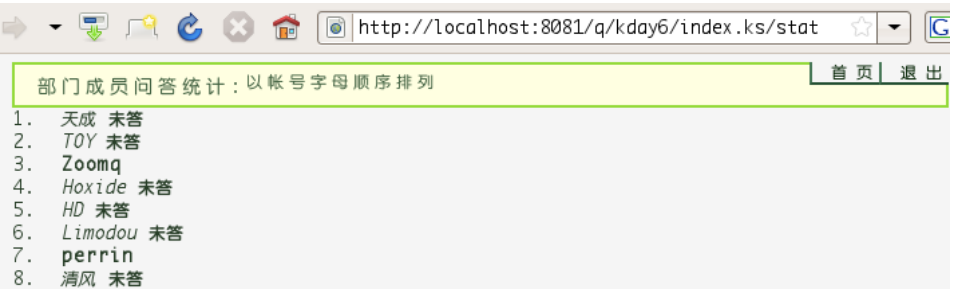


图 KDay6-7　成员回答情况统计汇报总表情景

最后就仅仅是信息的显示问题了。

1. 想要有回答时间的信息。

```
fn = conf.qpage.apath+'%s.%s.aq'%(qpname,a)
...
time.strftime("%y/%m/%d %H:%M:%S",time.localtime(os.path.getmtime(fn)
```

2. 想要有百分制的成绩。

```
 1 def _grade(right,answer):
```

```
    """根据问卷答案自动计算分数
    """
    grade = 0
    for i in range(0,len(right)):
        if right[i]==answer[i]:
            grade +=1
        else:
            pass
    return 100*(float(grade)/len(right))
```

唉呀？！小白突然发现，为什么不管怎么回答都是 0 分?

原来，是字典的无序和回答的有序之间的冲突造成的，验证一下是不是这个问题，如图 KDay6-8 所示：

```
print crtqp.ask.keys()
crtright = [crtqp.ask[i]["key"] for i in crtqp.ask.keys()]
print crtright
```



图 KDay6-8　将答案和问题教员检出到页面

知道症结就好修改了，如图 KDay6-9 所示。



图 KDay6-9

```
1 #字典排序技巧
2 ak = crtqp.ask.keys()
3 ak.sort()
4 print ak
5 crtright = [crtqp.ask[i]["key"] for i in ak]
6 print crtright
```

想要有总平均成绩，如图 KDay6-10 所示。

```
#使用 stat 列表收集所有有效成绩
stat = []
...
stat.append(_grade(crtright,open(fn,"r").read().split()))
...
sum(stat)/len(stat)) 就出来了。
或使用 reduce(lambda a，b：a+b，stat)/len(stat) 更加 cool!
```

哈哈哈!！什么也难不住小白了!



图 KDay6-10　有总平均分的情景

事务测试

不用想什么黑/白盒、边界，点击就成!好了!所有功能都实现了，小白非常有成就感地一路点击下来，感觉哪里不好，随手就加以修正它!溜个几回，功能测试基本也就 OK 了……

小结

小白在昨日基础上，使用 .ks 模式服务页面，将原先实现了的功能全部归并到了两个文

件中，进一步地完成以下任务：

1. 使用 .mm 文件作数据源来维护团队组织信息，并用包含的方式，应用到了页面事务中！
2. 发现了 JVF 的设计缺陷，并通过判定事务的迁移，通过服务端判别，重构了功能的支持！
3. 综合已有的成果，完成了统计功能的支持。

以下是本日成果：

```
kday5/
|-- Karrigell_QuickForm.py              KQF 快速表单模块
|-- dict4ini.py                         Limodou 贡献的 ini 解析模块
|-- a                                   回答收集目录
|   |-- easy051201.HD.aq                -实验问卷 HD 答案
|   |-- easy051201.Zoomq.aq             -实验问卷 Zoomq 答案
|   |-- easy051201.hoxide .aq           -实验问卷 hoxide 答案
|   `-- easy051201.perrin.aq            -实验问卷 perrin 答案
|-- q/                                  问卷设计文本收藏目录
|   |-- CPUG051211.cfg                  多问卷实例-CPyUG 社区问卷
|   |-- CPUG051211.cfg.051225182951
|   |-- Python051221.cfg                -Python 基础问卷
|   |-- Python051221.cfg.051225183000
|   |-- easy051201.cfg                  -实验问卷草稿
|   |-- easy051201.cfg.051222112545     -实验问题编辑历史快照
|   |-- ...
|   `-- easy051201.cfg.051225162724
|-- index.ks                            使用 KS 模式重构的有登录功能的系统首页
|-- mana.ks                             使用 KS 模式重构问卷设计综合支持页面
|-- mana.pih                            问卷设计页面
|-- qdesign.py                          问卷设计实际行为脚本
|-- qpage.pih                           问卷复审页面
|-- qpage.py                            问卷复审实际行为脚本
|-- qprint.pih                          问卷展示页面
|-- qprint.py                           问卷展示实际行为脚本
|-- questionnaire.cfg                   问卷系统配置文件
|-- questionnaire.tmpl                  问卷展示模板
|-- tryKQF.py                           KQF 试用页面
`-- xslmm                               freemind 思维图谱团队数据解析目录
   |-- deptorg.mm                       团队定义 freemind 底稿文件
   |-- deptorg.py                       团队定义 解析脚本
   |-- deptorg.xml                      团队定义 输出 XML 数据文件
```

```
|-- deptorganise.xsl                团队定义 输出 XSL 模板
|-- freemind.css                    freemind 输出默认样式
|-- freemind.xsl                    freemind 输出默认模板
|-- icons                           freemind 输出图标目录
|   |-- Mail.png
|   |-- ...
|   `-- xmag.png
|-- hide.png                        freemind 输出页面用小图形
|-- ...
`-- show.png
```

实例下载

实例下载请使用 SVN 下载地址。

下载地址: *http://openbookproject.googlecode.com/svn/trunk/LovelyPython/KDays/kday6*
精巧地址: *http://bit.ly/4phPmS*

练习

到目前为止，我们已经实现了登录、编辑、显示、删除、搜索博客文章，应该说作为个人博客已经差不多了，当然我们还能继续扩展，比如以下几点：

- 从单用户扩展为多个用户，变为一个简单的博客系统；
- 黑白页面比较丑，我们可以利用 css 做些页面的美化，比如加入颜色，布局调整等；
- 思考：前述字典排序是通过将字典关键词取出来列表排序的，有其他更加直接的方法吗？

KDayN　经验总结，畅想 Web 应用

坚持将自己很爽的体验整理为文章发布分享出来，不爽的也汇报回去，这些都是对社区的贡献!

小白将自己在问卷系统开发过程中的体验综合到了一张趋势图谱中（如图 KdayN-1 所示）：

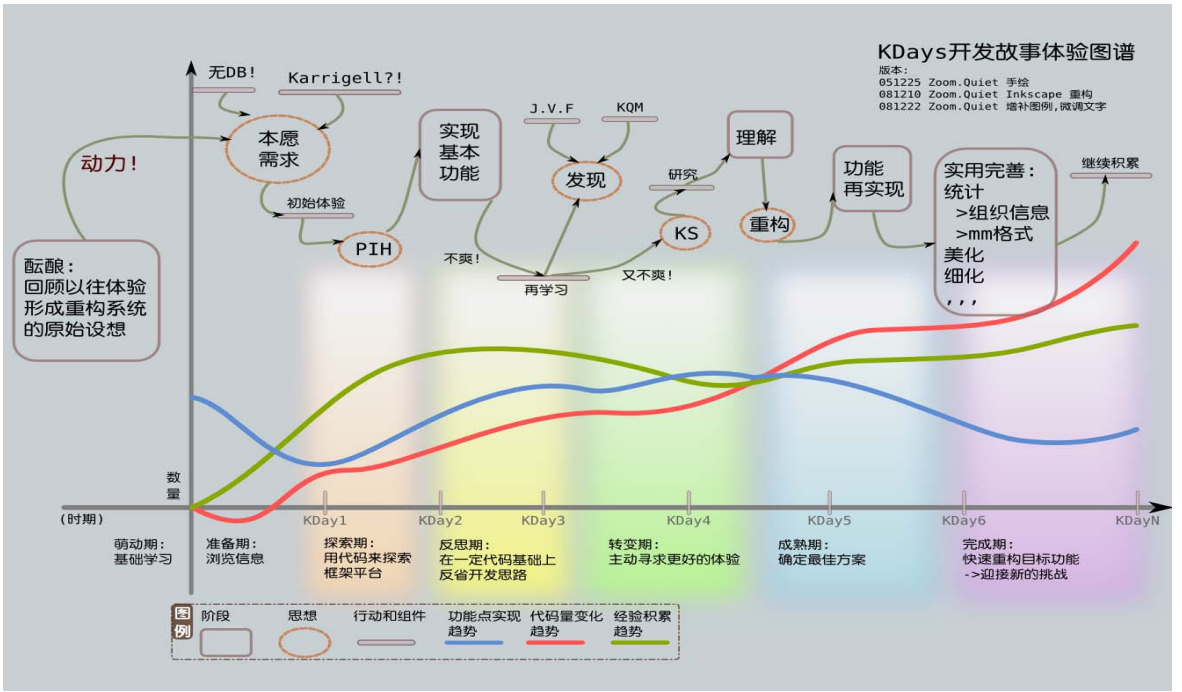


图 KdayN-1　问卷开发体验变迁趋势图

他进一步想到的优化方向是：

- 减少硬盘的读取次数；
- 减少内存占用空间；
- 加速响应处理；
- ……

这些是任何功能站点必定要面对的优化问题。

经验总结

在 Karrigell 中摸爬滚打几日，也算积累了一些经验，比如有关 Karrigell 开发的经验是：

- **教程要多看!**
- 开始在纯 .pih 模式中纠缠，回想起来真的累！要是早点对 .ks 的方便有所体验，就可以节省不少时间了。
- Cheetah 模板系统，已经在 Karrigell 2.2 系统中内含了，没有注意才使用了原始的方式。
- 充分信任 HTMLTags 函式化的 NHML 生成，感受就非常好!
- 勇于尝试，接受不同的思路。
- 同时也要勇于抛弃不好的代码。
- **只有更好，没有最好！！**保证当前已经完成的功能是可用的，是够用的。

有关 Karrigell 调试

页面空白，源代码都为空时，一定是 Python 崩溃，或是不吻合语法规则，根本没有运行起来！不过，Karrigell 足够皮实，不会崩溃的，马上按快捷键 `Ctrl+z` 几次，就可以知道哪里少括号什么的了。有时候 `print object` 页面没有任何变化，不要担心，看一下子 HTML 的源代码吧!

类似`<mod_ks.Script instance at 0x00E56DF0>` 这样的 Python 对象属性输出，在页面中会被浏览器处理为非法 HTML 节点而不显示的！

作为一个以快捷开发为豪的人，一定要不求甚解！好使就成，至于到底为什么，先别想，将来自然会明白的！

要知道人脑有下意识的，所以小白的所有疑问会在后台进程中一直运算到解答为止（即所谓**灵感**）。

有时候，明明是个字典对象，但是小白想不到根据相关键值来调用内容，那就使用.keys()[]的形式来输出吧；反正，小白只是要内容，不管键名的。

有关 Karrigell 站点组织

有关 Karrigell 站点组织，重构是必须的，自然的；同时**设计不如实现**。刚开始一定是模糊的遐想，只有快速变为可接触的功能，才可以进行改善，这样的开发才是最自然的。过程基本上是：

冲动→弱功能→利用已有的模块→可用→优化



如果牵连出三个以上的大问题的修改，立即换个方法！

在理性用户面前面向数据是核心！而且要连续作业，不然的话随意定的变量是干什么的，第二天一定想不起来。所以，也只有 Python 才有可能在少量的变量和代码中，实现小白的想象。

有关 CSS 设计

CSS 在提供了标准快捷的外观控制的同时也还有潜在问题。

在敏捷开发中，小白可以利用 CSS 快速改变外观，但是与语言不同的是，CSS 不能进行编辑判别，小白只能为不同的情况设立专用的 ID 或是类来协助 CSS 的渲染命中，导致 CSS 设定增长的速度比小白使用的函式还要快！可惜，现在没有什么好的想法来进行有效、聪明的控制。

TODO

至此，小白利用几天空闲时间，快速将原来以数据库为基础，PHP 为动态语言的**简单问卷系统**重构为纯 Python 实现，但是大多数功能都是随想随写成的，没有进行规划，要进一步开发的话，至少得有：

- 增加/删除问卷的功能；

- 丰富问卷形式，可以多选，填写补充信息；
- 支持 slide 方式，即一页接一页地回答；
- 丰富 JVF 的规则集合；
- 优化 JVF 的配置文件处理，将其嵌入到 HTML 的 `<XML>` 数据岛中；
- 优化 KQF 并与 JVF 配合起来，使之更加方便好用；
- 问卷系统外观风格的快速切换——实现皮肤功能。

……需求是变化莫测的，人的欲望也是无常的……但是，Python 的内涵是丰富的，功能是强大的，是可以快速支持所有想象的!

小白愿意分享 Pythonic 体验给任何有兴趣的读者，也热切期待更多热心人的好作品!

实例下载

同样，实例下载请使用 SVN 下载地址。

下载地址：*http://openbookproject.googlecode.com/svn/trunk/LovelyPython/KDays/kdayN*

精巧地址：*http://bit.ly/10qwJ3*

练习

读者做到这一步应该非常非常了解 Karrigell 的开发流程了，接下来还是要对已有的 Web 应用进行优化，这些只能通过自己不断的学习来逐步深入了。

Python 学习作弊条

Python 是活力四射的语言，是不断发展中的语言。就连使用 Python 多年的行者也不敢说对 Python 的方方面面都了解并可以自由运用，想必读者可能更加无法快速掌握所有重点技巧了。不过，不用沮丧，本书的观点一向是："用之！不学！"所以，行者根据自身体验，参照以往学校考试前加强记忆用的"作弊条"形式，组织了这部分的"Python 学习作弊条"，将本书实例故事讲述过程中涉及的相关知识点，以简洁易用的形式组织起来。一方面，给前述故事对应部分提供进一步解说；另一方面，也可以作为独立辞典快速查用。

【PCS 体例】

Python 学习作弊条（Python Cheat Sheet，PCS）使用统一的体例来记述知识要点：

概述 简洁地介绍知识点的来由。

使用 以实用的小例子，快速介绍知识点所涉及技术的应用方法。

问题 指出知识点应用中常常遇见的问题和解决思路。

探讨 进一步介绍相关或是深入的知识领域。

小结 总体回顾，给出相关阅读和思考指引。

相关练习和演示的代码，可以使用 SVN 下载到本地研究，下载地址：*http://openbookproject.googlecode.com/svn/trunk/LovelyPython/PCS/*

环境篇

PCS0 如何安装 Python 153
PCS1 交互环境之命令行 161
PCS2 交互环境之 iPython 164
PCS3 交互环境之 winpy 178
PCS4 常用自省 180
PCS5 Python 脚本文件 184
PCS6 Python 与中文 187
PCS7 Python 编码规范 191

PCS0　如何安装 Python

概述

Python 是跨平台的动态脚本语言，可以在 Windows、GNU/Linux、Unix、Mac OS X 等多种操作系统上运行，下面就 Python 在 Windows 和 GNU/Linux 上如何安装进行详细介绍。

应用

Windows 下安装 Python

在 Windows 下安装 Python 可以分为如下步骤。

1. 下载。从以下站点可以下载到 for Windows x86 的 Python2.5.2。

 1）Python 官方下载：*http://www.python.org/download/releases/2.5.2/*
 精巧地址：*http://bit.ly/1vlp1i*

 2）For Windows x86 的 Python2.5.2

 下载地址：*http://www.python.org/ftp/python/2.5.2/python-2.5.2.msi*
 精巧地址：*http://bit.ly/2uCvx5*

2. 下载完毕后，双击 python-2.5.2.msi 这个安装包，运行这个安装程序。出现安装提示“是为所有用户安装 Python，还是只为自己”，一般选使用系统默认选项“Install for all users”，也就是为所有用户安装 Python，直接点击“下一步”按钮，出现的界面

如图 PCS0-1 所示。

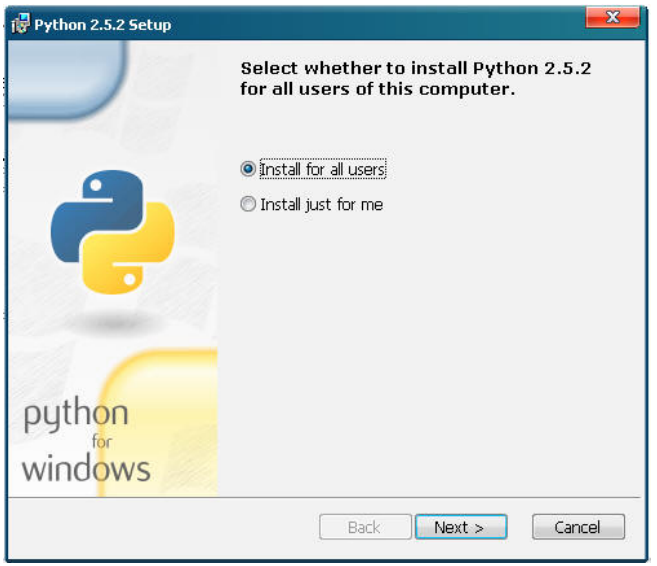


图 PCS0-1

3. 选择安装目录，Python 默认的安装目录是“C:\Python25”，可以直接使用默认目录安装，也可以安装到不同的目录。这里选择默认安装目录可直接点击“Next”按钮继续下面的安装，出现的界面如图 PCS0-2 所示。当然读者可以根据需要安装到其他目录。

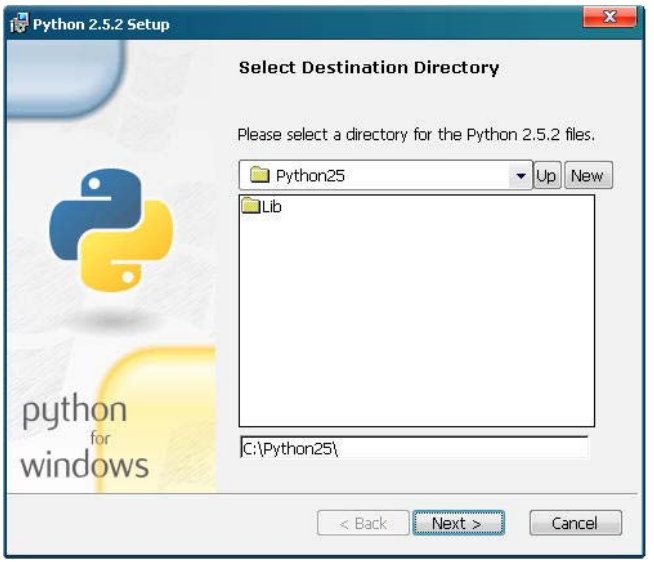


图 PCS0-2

4. 接下来，选择的是需要安装的 Python 组件，可以使用系统默认的选项全部安装，点击“Next”按钮进入下一步，如图 PCS0-3 所示。



图 PCS0-3

5. 出现进度条了（如图 PCS0-4），就表示开始安装 Python 了，需要等待一小会儿。

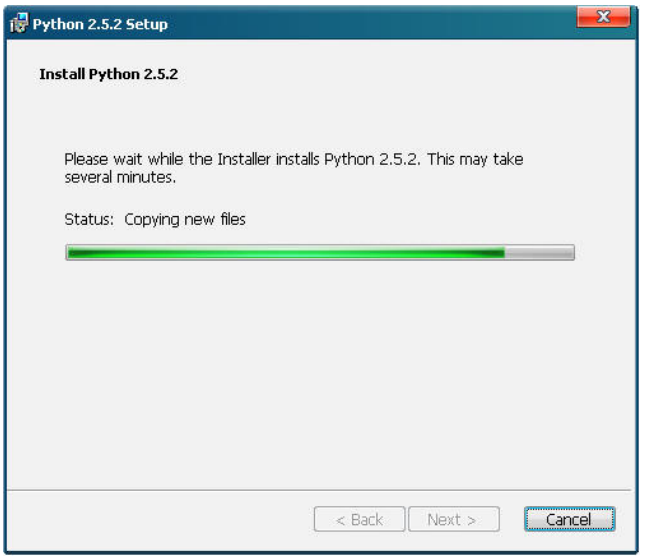


图 PCS0-4

6. 等待结束后，安装程序会提示 Python 安装完成（如图 PCS0-5），点击“Finish”按钮即可退出。



图 PCS0-5

这样，Windows 下的 Python 安装完毕，在 Windows 的“开始菜单”→“所有程序”中可找到 Python25，启动它即可使用。若想在 Windows 下的命令行中使用 Python，则还须要设置 Windows 的环境变量 Path。具体设置如下：右击“我的电脑”→选择“属性”→选择“高级”→点击“环境变量”，弹出环境变量对话框（如图 PCS0-6 所示），在系统变量中双击“Path”条目，弹出对话框，在变量值中加入路径为“C:\python25;”，注意这里的路径是之前安装 Python 时选择的安装目录。



图 PCS0-6

打开 Windows 的命令行，输入 python，即可进入 Python 交互环境（如图 PCS0-7 所示）。

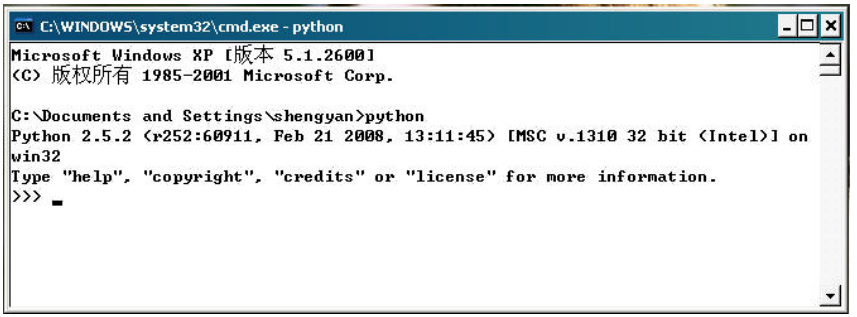


图 PCS0-7

GNU/Linux 下安装 Python

一般情况下，无须自己安装 Python。因为在大多数 GNU/Linux 版本中，如 Fedora、Ubuntu 等，都已经默认安装了 Python，但可以根据需要自定义安装 Python。下面使用源码编译安装来举个例子。

1. 下载源码包。这里下载的是 Python2.5.2 版本的 gzip 压缩包，读者也可以下载 Python2.5.2 版本的 bzip2 压缩包。

 （1）Python 官方下载：*http://www.python.org/download/releases/2.5.2/*
 精巧地址：*http://bit.ly/1vlp1i*

 （2）Python2.5.2 版本的 gzip 压缩包：*http://www.python.org/ftp/python/2.5.2/Python-2.5.2.tgz*
 精巧地址：*http://bit.ly/XlFop*

 （3）Python2.5.2 版本的 bzip2 压缩包：*http://www.python.org/ftp/python/2.5.2/Python-2.5.2.tar.bz2*
 精巧地址：*http://bit.ly/Inl29*

2. 解压安装包。在终端进入压缩包所在目录，输入命令 tar -zxvf Python-2.5.2.tgz (or bzcat Python-2.5.2.tar.bz2 | tar -xf -)，即可完成解压过程。

3. 进入解压后的 Python 目录，一般先看一下安装说明 README。这里详细讲述了 Python 2.5.2 的相关资源及其网址，以及各种支持的操作系统下的安装方法。读者可以根据需求适当参考该文档。下面给出 Python 最普通的安装过程。

4. ./configure（配置）。这里有个最常用设置的选项是 prefix，默认值为/usr/local/lib。设

置该选项来指定 Python 的安装目录，即./configure --prefix=$HOME/python2.5.2，$HOME 为用户主目录（如图 PCS0-8 所示）。

```
~/software$ cd Python-2.5.2/
~/software/Python-2.5.2$ ./configure --prefix=$HOME/python2.5.2
checking MACHDEP... linux2
checking EXTRAPLATDIR...
checking for --without-gcc... no
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
```

图 PCS0-8

5. make（编译源码）。若这里没出现什么错误即可进入下一步；若有错误，则可能有很多原因，比如当前系统缺少某些所需程序或尚未解决某些依赖关系，这样的话得一步一步地找出错误直至解决，才能编译正确。如图 PCS0-9 所示。

```
~/software/Python-2.5.2$ make
gcc -pthread -c -fno-strict-aliasing -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prot
otypes  -I. -IInclude -I./Include   -DPy_BUILD_CORE -o Modules/config.o Modules/
config.c
gcc -pthread -c -fno-strict-aliasing -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prot
otypes  -I. -IInclude -I./Include   -DPy_BUILD_CORE -DPYTHONPATH='":plat-linux2:
lib-tk"' \
                -DPREFIX='"/home/shengyan/python2.5.2"' \
                -DEXEC_PREFIX='"/home/shengyan/python2.5.2"' \
```

图 PCS0-9

6. make install（真正安装）。注意，若这里出现权限限制，则表明当前用户没有足够权限将 Python 程序文档文件写入指定的目录，比如说/usr/local/lib 等系统目录，这样的话，需要 sudo make install，输入密码后即可进行。因为之前设置了 prefix 为自己用户下的目录，所以直接 make install 就 ok 了。等待一段时间后，若没有错误提示就表明已经成功安装 Python2.5.2，界面如图 PCS0-10 所示。

```
~/software/Python-2.5.2$ make install
/usr/bin/install -c python /home/shengyan/python2.5.2/bin/python2.5
if test -f libpython2.5.so; then \
                if test ".so" = .dll; then \
                        /usr/bin/install -c -m 555 libpython2.5.so /home/shengya
n/python2.5.2/bin; \
                else \
                        /usr/bin/install -c -m 555 libpython2.5.so /home/shengya
n/python2.5.2/lib/libpython2.5.a; \
                        if test libpython2.5.so != libpython2.5.a; then \
                                (cd /home/shengyan/python2.5.2/lib; ln -sf libpy
```

图 PCS0-10

7. 成功安装后，在/home/计算机用户名/python2.5.2 下的就是刚刚安装的 Python 目录。在终端中进入/home/计算机用户名/python2.5.2 目录，输入./python，即可进入 Python 交互环境，可以看到刚才安装的 Python2.5.2（如图 PCS0-11）。

```
~$ cd python2.5.2/
~/python2.5.2$ cd bin
~/python2.5.2/bin$ ./python
Python 2.5.2 (r252:60911, Sep 27 2008, 18:55:47)
[GCC 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

图 PCS0-11

若想在终端中直接输入 python，启动的是 Python2.5.2，而不是以前的旧版本，则可以有以下两个方法实现：

方法一，修改环境变量 PATH。若仅仅在终端下修改 PATH，只对当前终端有效。若想在其他终端中或重启之后还有效的话，则须在用户目录的.bashrc 文件末尾加入 export PATH="/home/shengyan/python2.5/bin:$PATH"，注销或重启 X 即可，如图 PCS0-12 所示。具体设置 PATH 的多种方法可参考 *http://blog.csdn.net/wangyifei0822/archive/2008/05/04/2386076.aspx*（精巧地址：*http://bit.ly/4ncZ3U*）

```
~/python2.5.2/bin$ PATH=/home/shengyan/python2.5.2/bin:$PATH
~/python2.5.2/bin$ echo $PATH
/home/shengyan/python2.5.2/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
:/sbin:/bin:/usr/games
~/python2.5.2/bin$ python
Python 2.5.2 (r252:60911, Sep 27 2008, 18:55:47)
[GCC 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

图 PCS0-12

方法二，上述方法仅对当前用户环境进行设置，若想在系统上全局使用 Python2.5.2，则须进行以下步骤，如图 PCS0-13 所示。

```
~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
~$ which python
/usr/bin/python
~$ ls -l /usr/bin/python
lrwxrwxrwx 1 root root 9 2008-06-02 04:21 /usr/bin/python -> python2.5
~$ sudo mv /usr/bin/python /usr/bin/python_2.5.1
~$ sudo ln -s /home/shengyan/python2.5.2/bin/python /usr/bin/python
~$ python
Python 2.5.2 (r252:60911, Sep 27 2008, 18:55:47)
[GCC 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

图 PCS0-13

小结

本文详细介绍了在 Windows 和 GNU/Linux 下安装 Python 的方法。更多关于 Python 的介

绍及其安装可到以下网站查看：

（1）Python 官方英文网站：*http://www.python.org/*

（2）Python 中文社区：*http://python.cn/*

（3）Python 英文官方文档及资源：*http://www.python.org/doc/*

（4）Python 手册：*http://doc.chinahtml.com/Manual/Python/tut/index.html*

精巧地址：*http://bit.ly/XbkRB*

PCS1　交互环境之命令行

概述

Python 命令行，又称为 Python Shell，是默认的 Python 交互环境。

应用

进入 Python Shell

若在 Windows 下已经安装好 Python，也设置好了相应环境变量。在 GNU/Linux 下，通常情况是已经安装好 Python 了的，默认安装在/usr/bin/python 下，该路径已经放进你的 shell 搜索路径中。

打开 Windows 的命令行或 GNU/Linux 的终端，输入 python，即可进入 Python 交互式环境，界面如图 PCS1-1 所示。

```
~$ python
Python 2.5.2 (r252:60911, Sep 27 2008, 18:55:47)
[GCC 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'hi'
hi
>>>
```

图 PCS1-1

进入后，可以编写代码，调试，测试及查看相关帮助。若所做工作完成，退出命令行，

可以使用 Ctrl+Z+Enter 键（Windows 下）或 Ctrl+D（GNU/Linux 下）键以 0 值退出（就是说，没有什么错误，正常退出）。如果这没有起作用，可以输入以下命令退出：import sys; sys.exit()。

使用交互环境

进入 Python Shell 后（如图 PCS1-1 所示），我们来具体介绍如何使用该环境。

Python 解释器根据主提示符来执行，主提示符通常标识为三个“大于”号（>>>）；继续的部分被称为从属提示符，由三个点标识（...）。在第一行之前，解释器打印“欢迎信息、版本号和授权”，提示如下：

```
Python 2.5.1 (r251:54863, Mar  7 2008, 04:10:12)
[GCC 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

输入多行结构时须用到从属提示符了。例如，下面这个 if 语句：

```
>>> sayhi = True
>>> if sayhi:
...     print 'hi~python!'
... else:
...     print 'say nothing!'
...
hi~python!
```

若在调试使用过程中有错误发生，则解释器会打印一个报错信息、栈跟踪器及出错位置等，便于修改。此为错误处理。

```
>>> if sayhi:
...     print 'hi~python!'
... else:
...
  File "<stdin>", line 4

    ^
IndentationError: expected an indented block
>>> import Mysqldb
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named Mysqldb
>>>
```

若在主提示符或附属提示符中输入中断符（Control-c，抛出一个 keyboardinterrupt 异常，它可以被 try 句截获）就会取消当前输入，回到主命令行。

在 Python Shell 中可以很方便地查看 Python 文档，包括类型、类库、模块等的使用资料。这些都是非常有用的。通过 help（"obj"）就可以看到 obj 的帮助信息，就像系统 Shell 中的 man 帮助一样，它提供了非常详细的资料。

小结

本文讲述了最基本的 Python 命令行使用方式，并描述了如何进入、使用、退出交互环境。这是非常简易的，很多情况下可以先在这里做些代码测试，通过后再进行脚本编辑。

练习

Python 是解释执行的，那么什么是解释执行呢？和其他语言（Java、C 等）的执行方式有什么区别？

相关参考

使用 Python 解释器：*http://doc.chinahtml.com/Manual/Python/tut/node4.html*

PCS2　交互环境之 iPython

概述

iPython 是一个 Python 的交互式 Shell，比默认的 Python Shell 好用得多，功能也更强大。她支持语法高亮、自动完成、代码调试、对象自省，支持 Bash Shell 命令，内置了许多很有用的功能和函式等，非常容易使用。

应用

目前，最新稳定的 iPython 版本是 0.8.4 版，支持多种操作系统，如 GNU/Linux、Unix、Mac OS X、Windows 等，这里详细介绍在 GNU/Linux 和 Windows 下的安装过程。其他系统下的安装过程可参见 *http://ipython.scipy.org/moin/Download*（精巧地址：*http://bit.ly/1sRzxv*）上的具体说明。

Windows 下的 iPython 安装

在 Windows 下安装 iPython 可分为以下几步：

1. 下载 ipython-0.8.4.win32-setup.exe 和 pyreadline-1.5-win32-setup.exe。

 （1）下载 ipython-0.8.4.win32：*http://ipython.scipy.org/dist/ipython-0.8.4.win32-setup.exe*
 精巧地址：*http://bit.ly/YqbkJ*

（2）下载 pyreadline-1.5-win32：*http://ipython.scipy.org/dist/pyreadline-1.5-win32-setup.exe*
精巧地址：*http://bit.ly/2JFKDM*

2. 双击运行安装程序，只需经过 4 步即可完成安装。

第一步，显示一些提示信息如图 PCS2-1，直接点击“下一步”按钮。

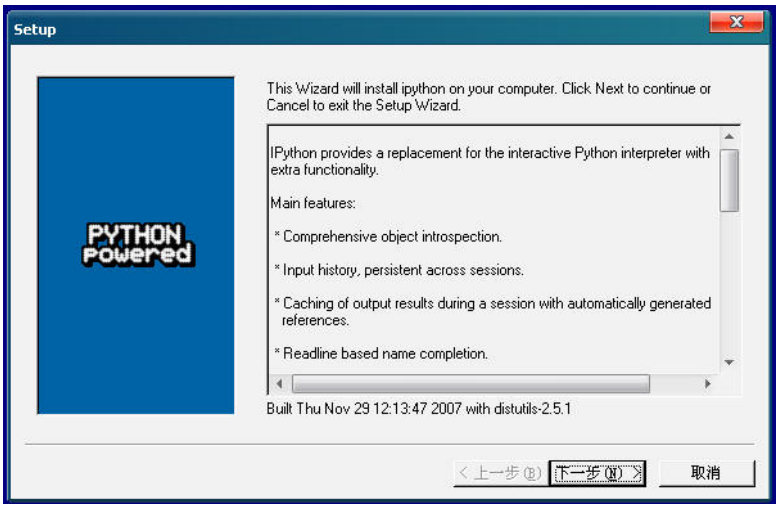


图 PCS2-1

第二步，输入 Python 安装目录及 iPython 安装位置，因为之前把 Python 默认安装在“C:\Python25\”了，所以这里两个都为默认设置（如图 PCS2-2 所示），直接点击“下一步”按钮。若之前 Python 不是在“C:\Python25\”，则需要改变为 Python 所在的目录。

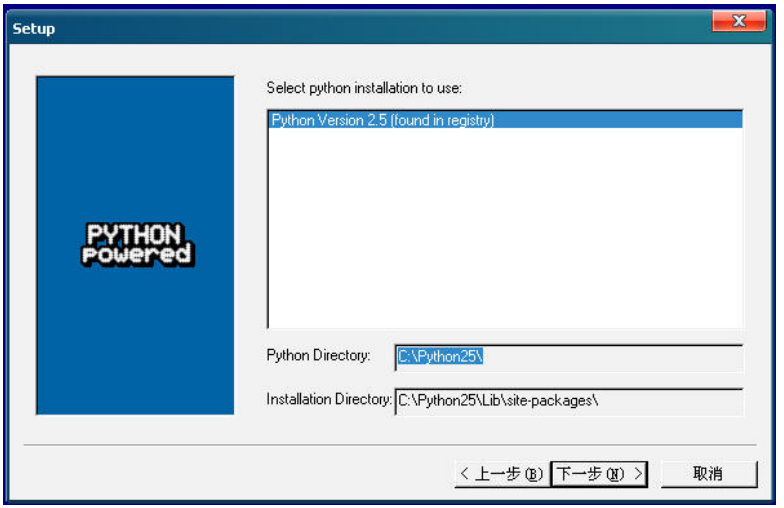


图 PCS2-2

第三步，出现进度条，如图 PCS2-3 所示。须等待一会儿，完毕后点击“下一步”按钮。

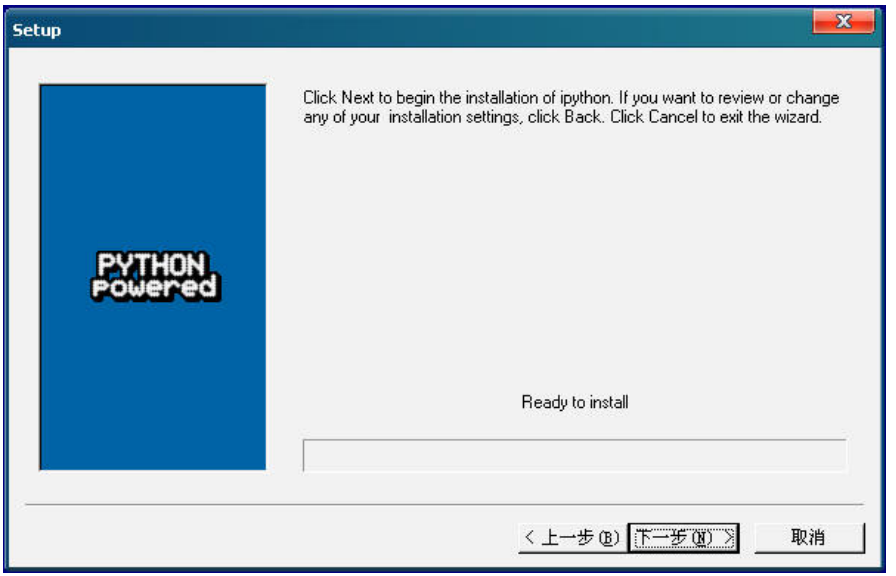


图 PCS2-3

第四步，显示完成（如图 PCS2-4 所示），成功安装 iPython。

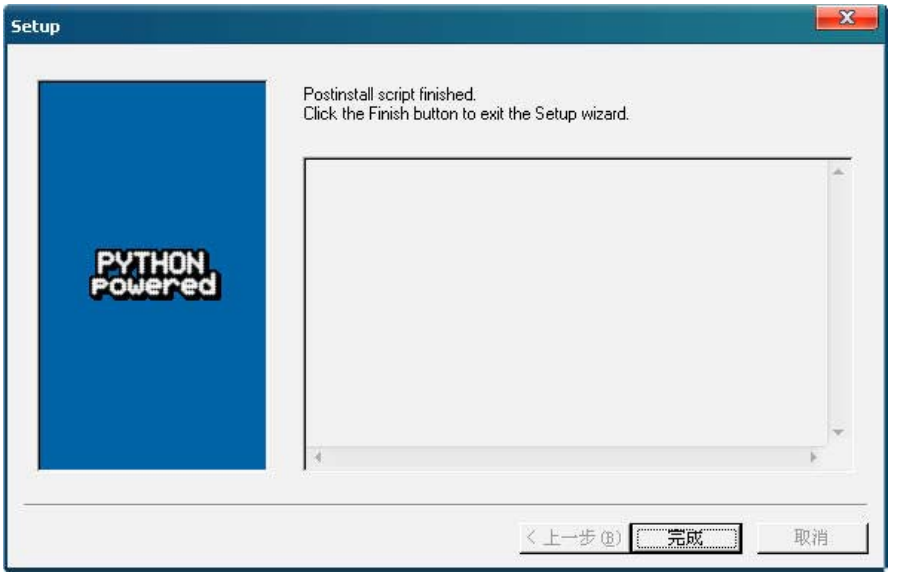


图 PCS2-4

3. 安装 pyreadline。直接双击安装，如图 PCS2-5 所示。

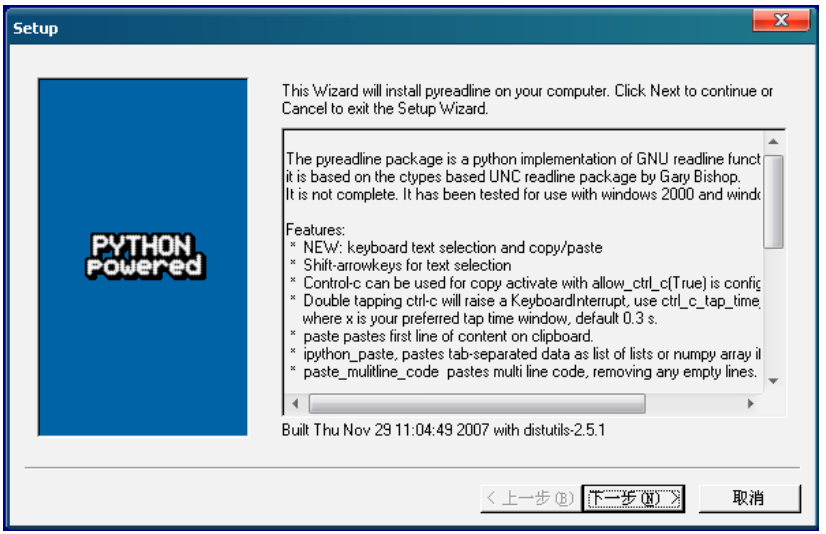


图 PCS2-5

接下去，只须点击“下一步”按钮直至安装完毕即可。

4. 设置环境变量 Path，在它里面加上 Python 安装目录下面的 scripts 目录，如图 PCS2-6 所示。

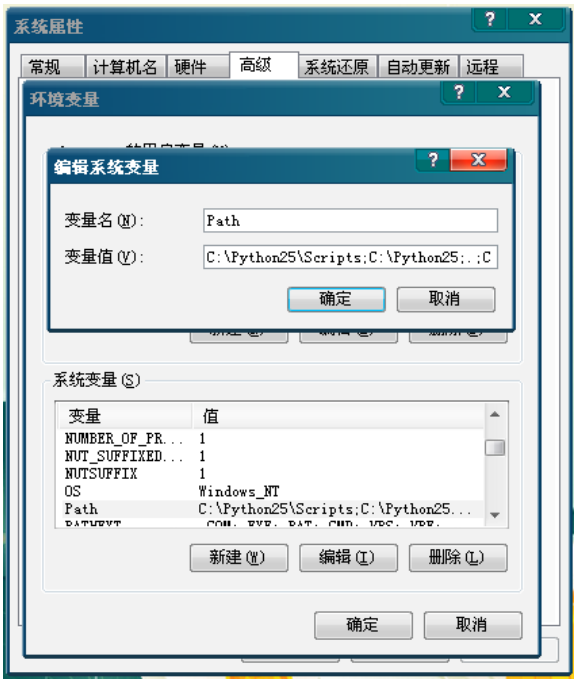


图 PCS2-6

5. 安装完成后，进入 Windows 命令行，输入 ipython，即可以进入 iPython 交互环境如图 PCS2-7 所示。

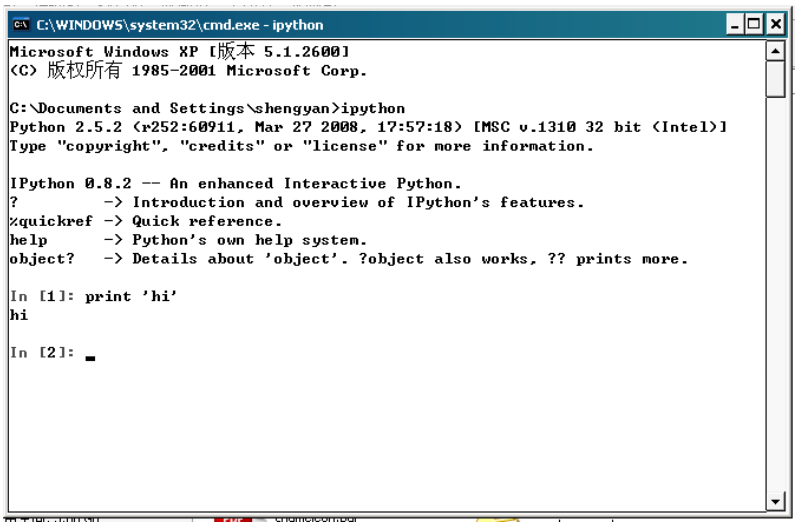


图 PCS2-7

GNU/Linux 下的 iPython 安装

1. 下载 ipython-0.8.4.tar.gz。

 （1）下载 ipython-0.8.4.tar.gz 包：*http://ipython.scipy.org/dist/ipython-0.8.4.tar.gz*

 精巧地址：*http://bit.ly/1VaGJw*

2. 进入终端，输入如下几条命令，分别是解压、进入目录、编译、安装，一切 ok 的话就顺利安装了 iPython。

```
~$ tar xvzf ipython-0.8.4.tar.gz
~$ cd ipython-0.8.4
~/ipython-0.8.4$ python setup.py build  #这步可以省略
~/ipython-0.8.4$ sudo python setup.py install
```

3. 在 Ubuntu 下，输入命令 sudo apt-get install ipython 即可完成安装，非常方便。

使用 iPython

在顺利安装好 iPython 之后，就可以进入该交互环境使用了。下面介绍在 GNU/Linux 下

iPython 的使用方式，Windows 下也是很类似的。下面的内容很多参考自：

使用 iPython 增强交互式体验：*http://forum.ubuntu.org.cn/viewtopic.php?p=447255&sid=5ba2eaa5af49eaca994976f9c285b819*

精巧地址：*http://bit.ly/1v24Sl*

其英文原文“Enhanced Interactive Python with iPython”：*http://www.onlamp.com/pub/a/python/2005/01/27/ipython.html*

精巧地址：*http://bit.ly/1h699v*

非常不错的资料！

进入终端，输入 ipython，即可进入如图 PCS2-8 所示的 iPython 交互环境。

```
shengyan@LIZZIE:~/openbookprojectnew$ ipython
Python 2.5.2 (r252:60911, May  7 2008, 15:19:09)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.4 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

In [1]: print 'hi'
hi

In [2]:
```

图 PCS2-8

注意：若是第一次运行 ipython 时会自动生成配置目录 $HOME/.ipython，它里面包含的一些配置文件适用于不同的环境：ipythonrc、ipythonrc-math、ipythonrc-numeric、ipythonrc-physics、ipythonrc-pysh、ipythonrc-scipy 及 ipythonrc-tutorial。先可以不用管它，直接使用默认就好了。

在交互环境和在 Python 默认交互环境中一样，编写代码进行调试、测试等。但比默认 Python 环境好的有如下几点。

1. Magic。iPython 有一些“magic”关键字：

```
%Exit, %Pprint, %Quit, %alias, %autocall, %autoindent, %automagic,
%bookmark, %cd, %color_info, %colors, %config, %dhist, %dirs, %ed,
%edit, %env, %hist, %logoff, %logon, %logstart, %logstate, %lsmagic,
%macro, %magic, %p, %page, %pdb, %pdef, %pdoc, %pfile, %pinfo, %popd,
%profile, %prun, %psource, %pushd, %pwd, %r, %rehash, %rehashx, %reset,
%run, %runlog, %save, %sc, %sx, %system_verbose, %unalias, %who,
%who_ls, %whos, %xmode
```

iPython 会检查传给它的命令是否包含 magic 关键字。如果命令是一个 magic 关键字，iPython 就自己来处理。如果不是 magic 关键字，就交给 Python 去处理。如果 automagic 打开（默认），不须要在 magic 关键字前加%符号。相反，如果 automagic 是关闭的，则%是必需的。在命令提示符下输入命令 magic 就会显示所有 magic 关键字列表，以及它们简短的用法说明。良好的文档对于一个软件的任何一部分来说都是重要的，从在线 iPython 用户手册到内嵌文档（%magic），iPython 当然不会在这方面有所缺失。下面介绍些常用的 magic 函式，如：

```
%bg function
    把 function 放到后台执行，例如：%bg myfunc(x, y, z=1)，之后可以用 jobs 将其结果取回，myvar = jobs.result(5)或 myvar = jobs[5].result。另外，jobs.status()可以查看现有任务的状态。
%ed 或 %edit
    编辑一个文件并执行，如果只编辑不执行，用 ed -x filename 即可。
%env
    显示环境变量
%hist 或 %history
    显示历史记录
%macro name n1-n2 n3-n4 ... n5 .. n6 ...
    创建一个名称为 name 的宏，执行 name 就是执行 n1-n2 n3-n4 ... n5 .. n6 ... 这些代码。
%pwd
    显示当前目录
%pycat filename
    用语法高亮显示一个 Python 文件（不用加.py 后缀名）
%save filename n1-n2 n3-n4 ... n5 .. n6 ...
    将执行过多代码保存为文件
%time statement
    计算一段代码的执行时间
%timeit statement
    自动选择重复和循环次数计算一段代码的执行时间，太方便了。
```

2. iPython 中用！表示执行 shell 命令，用$将 Python 的变量转化成 Shell 变量。通过这两个符号，就可以做到和 Shell 命令之间的交互，可以非常方便地做许多复杂的工作。比如可以很方便地创建一组目录：

```
for i in range(10):
    s = "dir%s" % i
    !mkdir $s
```

不过写法上还是有一些限制，$ 后面只能跟变量名，不能直接写复杂表达式，$"dir%s"%i 就是错误的写法了，所以要先完全产生 Python 的变量以后再用。例如：

```
for i in !ls:
   print i
```

这样的写法也是错的，可以这样：

```
a = !ls
for i in a:
   print i
```

还有一点需要说明，就是执行普通的 Shell 命令中如果有 $ 的话需要用两个 $。比如原来的 echo $PATH 现在得写成!echo $$PATH。

3. **Tab 自动补全**。iPython 一个非常强大的功能是 tab 自动补全。标准 Python 交互式解释器也可以 tab 自动补全：

```
~$ python
Python 2.5.1 (r251:54863, Mar  7 2008, 04:10:12)
[GCC 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import rlcompleter, readline
>>> readline.parse_and_bind('tab: complete')
>>> h
hasattr  hash    help    hex
```

标准 Python 交互式解释器和 iPython 都支持“普通”自动补全和菜单补全。

使用自动补全，要先输入一个匹配模型，然后按 Tab 键。如果是“普通”自动补全模式（默认），按 Tab 键后会：

- 匹配模型按最大匹配展开；
- 列出所有匹配的结果。

例如：

```
In [1]: import os

In [2]: os.po
os.popen os.popen2 os.popen3 os.popen4

In [2]: os.popen
```

输入 os.po 然后按 Tab 键，os.po 被展开成 os.popen（就像在 In [2]:提示符显示的那样），并显示 os 所有以 po 开头的模块、类和函式，它们是 popen、popen2、popen3 和 popen4。

而菜单补全稍有不同。关闭默认 Tab 补全，使用菜单补全，须修改配置文件 $HOME/.ipython/ipythonrc。

注释掉：

```
readline_parse_and_bind tab: complete
```

取消注释：

```
readline_parse_and_bind tab: menu-complete
```

不同于“普通”自动补全的显示，当前命令所有匹配列表的菜单补全会随着每按一次 Tab 键而循环显示匹配列表中的项目。例如：

```
In [1]: import os

In [2]: os.po
```

结果是：

```
In [3]: os.popen
```

接下来每次按 Tab 键就会循环显示匹配列表中的其他项目：popen2、popen3、popen4，最后回到 po。菜单补全模式下查看所有匹配列表的快捷键是 Ctrl+L。

4. **自省**。Python 有几个内置的函式用于自省。iPython 不仅可以调用所有标准 Python 函式，对于那些 Python shell 内置函式同样适用。典型的使用标准 Python shell 进行自省是使用内置的 dir()函式：

```
>>> import SimpleXMLRPCServer
>>> dir(SimpleXMLRPCServer.SimpleXMLRPCServer)
['__doc__', '__init__', '__module__', '_dispatch',
'_marshaled_dispatch', 'address_family', 'allow_reuse_address',
'close_request', 'fileno', 'finish_request', 'get_request',
'handle_error', 'handle_request', 'process_request',
'register_function', 'register_instance',
'register_introspection_functions', 'register_multicall_functions',
'request_queue_size', 'serve_forever', 'server_activate', 'server_bind',
'server_close', 'socket_type', 'system_listMethods',
'system_methodHelp', 'system_methodSignature', 'system_multicall',
'verify_request']
```

因为 dir()是一个内置函式，在 iPython 中也能很好地使用它们。但是 iPython 的操作符?和??功能还要强大：

```
In [3]: import SimpleXMLRPCServer

In [4]: ? SimpleXMLRPCServer
Base Class:      <type 'module'>
String Form: <module 'SimpleXMLRPCServer' from
             '/usr/lib/python2.5/SimpleXMLRPCServer.pyc'>
Namespace:       Interactive
File:           /usr/lib/python2.5/SimpleXMLRPCServer.py
```

```
Docstring:
   Simple XML-RPC Server.

   This module can be used to create simple XML-RPC servers
   by creating a server and either installing functions, a
   class instance, or by extending the SimpleXMLRPCServer
   class.

   It can also be used to handle XML-RPC requests in a CGI
   environment using CGIXMLRPCRequestHandler.

   A list of possible usage patterns follows:

   1. Install functions:

   server = SimpleXMLRPCServer(("localhost", 8000))
   server.register_function(pow)
   server.register_function(lambda x,y: x+y, 'add')
:
```

?操作符会截断长的字符串。相反，??不会截断长字符串，如果有源代码的话还会以语法高亮形式显示它们。

5. **历史**。当在 iPython shell 下交互地输入了大量命令、语句等，就像这样：

```
In [1]: a = 1

In [2]: b = 2

In [3]: c = 3

In [4]: d = {}

In [5]: e = []

In [6]: for i in range(20):
   ...:     e.append(i)
   ...:     d[i] = b
   ...:
```

可以输入命令“hist”快速查看那些已输入的历史记录：

```
In [7]: hist
1: a = 1
2: b =2
```

```
3: c = 3
4: d = {}
5: e = []
6:
for i in range(20):
    e.append(i)
    d[i] = b

7: _ip.magic("hist ")
```

要去掉历史记录中的序号（这里是 1 至 7），可使用命令“hist -n”：

```
In [8]: hist -n
a = 1
b =2
c = 3
d = {}
e = []
for i in range(20):
    e.append(i)
    d[i] = b

_ip.magic("hist ")
_ip.magic("hist -n")
```

这样就可方便地将代码复制到一个文本编辑器中。要在历史记录中搜索，可以先输入一个匹配模型，然后按 Ctrl+P 键。找到一个匹配后，继续按 Ctrl+P 键就会向后搜索再上一个匹配，按 Ctrl+N 键则是向前搜索最近的匹配。

6. **编辑**。如果想在 Python 提示符下试验一个想法，经常要通过编辑器修改源代码（甚至是反复修改）。在 iPython 下输入 edit 就会根据环境变量$ EDITOR 调用相应的编辑器。如果$EDITOR 为空，则会调用 vi（Unix）或记事本（Windows）。要回到 iPython 提示符，直接退出编辑器即可。如果是保存并退出编辑器，输入编辑器的代码会在当前名字空间下被自动执行。如果不想这样，可使用 edit+X。如果要再次编辑上次最后编辑的代码，使用 edit+P。在上一个特性里，提到使用 hist–n 可以很容易地将代码拷贝到编辑器。一个更简单的方法是 edit 加 Python 列表的切片（slice）语法。假定 hist 输出如下：

```
In [29]: hist
1 : a = 1
2 : b = 2
3 : c = 3
4 : d = {}
```

```
5 : e = []
6 :
for i in range(20):
e.append(i)
d[i] = b

7 : %hist
```

现在要将第 4、5、6 句代码导出到编辑器，只要输入：

```
edit 4:7
```

7. `Debugger` 接口。iPython 的另一特性是它与 Python debugger 的接口。在 iPython Shell 下输入 magic 关键字 pdb 就会在产生一个异常时开关自动 debugging 功能。在 pdb 自动呼叫启用的情况下，当 Python 遇到一个未处理的异常时 Python debugger 就会自动启动。debugger 中的当前行就是异常发生的那一行。iPython 的作者说有时候当他需要在某行代码处 debug 时，他会在开始 debug 的地方放一个表达式 1/0。启用 pdb，在 iPython 中运行代码。当解释器处理到 1/0 那一行时，就会产生一个 ZeroDivisionError 异常，然后它就从指定的代码处被带到一个 debugging session 中了。

8. 运行。有时候在一个交互式 Shell 中，如果可以运行某个源文件中的内容将会很有用。运行 magic 关键字 run 带一个源文件名就可以在 iPython 解释器中运行一个文件了（例如 run <源文件> <运行源文件所需参数>）。参数主要有以下这些：

 - -n 阻止运行源文件代码时`__name__`变量被设为`"__main__"`。这会防止 `if __name__ == "__main__":`块中的代码被执行。
 - -i 源文件在当前 iPython 的名字空间下运行而不是在一个新的名字空间中。如果你需要源代码可以使用在交互式 session 中定义的变量，它会很有用。
 - -p 使用 Python 的 profiler 模块运行并分析源代码。使用该选项的代码不会运行在当前名字空间。

9. 宏。宏允许用户为一段代码定义一个名字，这样可在以后使用这个名字来运行这段代码。就像在 magic 关键字 edit 中提到的，列表切片法也适用于宏定义。假设有一个历史记录如下：

```
In [3]: hist
1: l = []
2:
for i in l:
print i
```

可以这样来定义一个宏：

```
In [4]: macro print_l 2
```

```
Macro `print_l` created. To execute, type its name (without quotes).
Macro contents:
for i in l:
print i
```

运行宏：

```
In [5]: print_l
------> print_l()
```

在这里，列表 l 是空的，所以没有东西被输出。但这其实是一个很强大的功能，赋予列表 l 某些实际值，再次运行宏就会看到不同的结果：

```
In [7]: l = range(5)

In [8]: print_l
------> print_l()
0
1
2
3
4
```

当运行一个宏时就好像你重新输入了一遍包含在宏 print_l 中的代码。它还可以使用新定义的变量 l。由于 Python 语法中没有宏结构（也许永远也不会有），在一个交互式 shell 中它更显得是一个有用的特性。

10. **环境**（`Profiles`）。就像早先提到的那样，iPython 安装了多个配置文件用于不同的环境。配置文件的命名规则是 ipythonrc-。要使用特定的配置启动 iPython，需要这样：

```
ipython –p
```

一个创建自己环境的方法是在$HOME/.ipython 目录下创建一个 iPython 配置文件，名字就叫做 ipythonrc-，这里是你想要的环境名字。如果同时进行好几个项目，而这些项目又用到互不相同的特殊的库，这时候每个项目都有自己的环境就很有用了。也可以为每个项目建立一个配置文件，然后在每个配置文件中 import 该项目中经常用到的模块。

11. **使用操作系统的** `Shell`。使用默认的 iPython 配置文件，有几个 Unix Shell 命令（当然，是在 Unix 系统上），cd、pwd 和 ls 都能像在 bash 下一样工作。运行其他的 shell 命令须要在命令前加!或!!。使用 magic 关键字%sc 和%sx 可以捕捉 shell 命令的输出。pysh 环境可以被用来替换掉 shell。使用-p pysh 参数启动的 iPython，可以接受并执行用户$PATH 中的所有命令，同时还可以使用所有的 Python 模块、Python 关键字和

内置函式。例如，想要创建 500 个目录，命名规则是从 d_0_d 到 d_499_d，可以使用 -p pysh 启动 iPython，然后就像这样：

```
[~/ttt]|1> for i in range(500):
                       |.>    mkdir d_${i}_d
                       |.>
```

这就会创建 500 个目录：

```
[~/ttt]|2> ls -d d* | wc -l
500
```

注意这里混合了 Python 的 range 函式和 Unix 的 mkdir 命令。虽然 ipython -p pysh 提供了一个强大的 shell 替代品，但它缺少正确的 job 控制。在运行某个很耗时的任务时按下 Ctrl+Z 键将会停止 iPython session 而不是那个子进程。

最后，退出 iPython。可输入 Ctrl+D 键（会要求你确认），也可以输入 Exit 或 Quit（注意大小写）退出而无须确认。

小结

经过本文对 iPython 的特性及其基本使用方法的介绍，已经充分感受到 iPython 的强大功能了吧！那么，就把它作为一个有利的工具帮助我们开发吧！对于进一步的配置和更多的用途，有兴趣的读者可以在以下网络上发掘更丰富的资料。

- iPython 网站：*http://ipython.scipy.org*
- iPython 英文文档：*http://ipython.scipy.org/moin/Documentation*
 精巧地址：*http://bit.ly/GdPZU*
- iPython 中一些 magic 函式：*http://guyingbo.javaeye.com/blog/111142*
 精巧地址：*http://bit.ly/3GVxE8*

PCS3　交互环境之 winpy

概述

ActivePython 是标准的 Python 发行版，在 Windows、GNU/Linux、Mac OS X 等平台上都能使用。

应用

这里只介绍 ActivePython 在 Windows 下的安装：

1. 下载 ActivePython。

 ActivePython 官方下载：*http://downloads.activestate.com/ActivePython/windows/2.5/ActivePython-2.5.2.2-win32-x86.msi*

 精巧地址：*http://bit.ly/3UaHr8*

2. 双击安装文件，运行安装程序。该过程和很多软件在 Windows 下的安装很类似，在这里就不描述了。主要应注意的地方是在 ActivePython 安装时须选对 Python 路径（这表示事先得安装好 Python），其他都可以默认。

3. 安装完毕。打开 ActivePython 的 PythonWin，可以看到如图 PCS3-1 所示的界面。

其中包含的一个交互界面的使用方式和 Python 默认环境非常类似。

交互界面使用方式具体可参考 PCS1

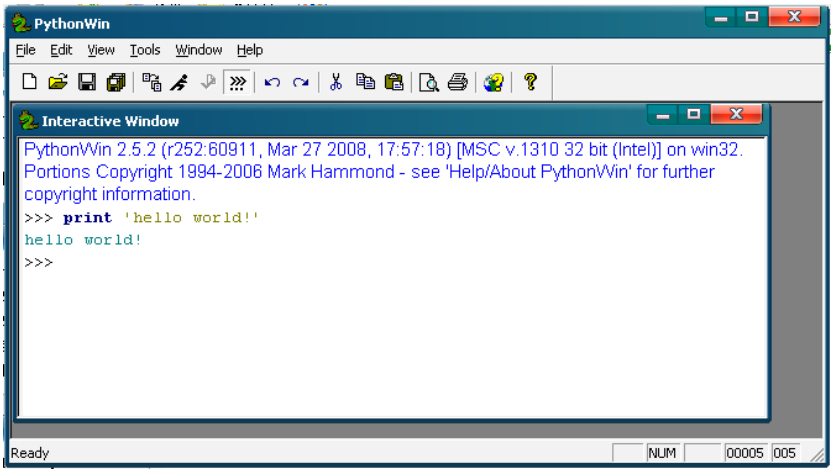


图 PCS3-1

小结

ActivePython 是一个不错的 Python 编辑器和运行调试工具。界面比较简单，但功能对于初学者来说已经足够了。

PCS4　常用自省

概述

自省，简单来说就是自我检查。Python 提供这个强大的功能，以方便程序员查看各个对象的信息。常用自省函式有 `help()`、`dir()`、`type()`、`id()`等。

应用

help()

用来查看很多 Python 自带的帮助文档信息，可以使用 `help()`。

```
~$ ipython
Python 2.5.2 (r252:60911, May  7 2008, 15:19:09)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.2 -- An enhanced Interactive Python.
?        -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

In [1]: help("str")
```

```
In [2]: import pygments

In [3]: help("pygments")
```

如上所示，在 Python 交互环境中，输入 `help("obj")`后即可看到 obj 的帮助信息。就像 bash 中 man 随机帮助页一样。另外，也可以直接通过 `help()`进入如下帮助实用程序：

```
In [4]: help()

Welcome to Python 2.5!  This is the online help utility.

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://www.python.org/doc/tut/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules",
"keywords", or "topics".  Each module also comes with a one-line summary
of what it does; to list the modules whose summaries contain a given word
such as "spam", type "modules spam".

help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

and             elif            if              print
as              else            import          raise
assert          except          in              return
break           exec            is              try
class           finally         lambda          while
continue        for             not             with
def             from            or              yield
del             global          pass
```

dir()

`dir()`可以列出对象的所有属性，如下：

```
In [6]: import os
```

```
In [7]: dir(os)
Out[7]:
['EX_CANTCREAT',
 'EX_CONFIG',
 'EX_DATAERR',
 'EX_IOERR',
 'EX_NOHOST',
 'EX_NOINPUT',
 ......
```

type()

`type()`返回对象的类型，如下：

```
In [8]: type(os)
Out[8]: <type 'module'>
```

id()

`id()`返回对象的“唯一序号”。其中，对于引用对象来说，返回的是被引用对象的 `id()`。

```
In [29]: a = 'abc'

In [30]: b = a

In [31]: id(a)
Out[31]: 137472992

In [32]: id(b)
Out[32]: 137472992
```

hasattr()和 getattr()

`hasattr()`和 `getattr()`分别判断对象是否有某个属性及获得某个属性值。

```
In [35]: hasattr(a, 'split')
Out[35]: True

In [36]: getattr(a, 'split')
Out[36]: <built-in method split of str object at 0x831abe0>
```

callable()

`callable()`判断对象是否可以被调用。

```
In [37]: callable(a)
Out[37]: False

In [38]: callable(a.split)
Out[38]: True
```

isinstance()

`isinstance()`可以确认某个变量是否有某种类型。

```
In [44]: isinstance(a, str)
Out[44]: True

In [45]: isinstance(a, int)
Out[45]: False
```

小结

以上这些都是常用的自省方法，还有其他的可参见相关资源：

- 自省的威力：

 http://www.woodpecker.org.cn/diveintopython/power_of_introspection/index.html

 精巧地址：*http://bit.ly/2cYq5p*

- Python 自省指南：*http://www.ibm.com/developerworks/cn/linux/l-pyint/index.html*

 精巧地址：*http://bit.ly/32ZEgp*

- Python 学习笔记- 2.自省：*http://www.xwy2.com/article.asp?id=106*

 精巧地址: *http://bit.ly/J4LZ5*

PCS5　Python 脚本文件

概述

脚本一般都是以文本形式存在的文件。它不无须像 C 语言那样编译成二进制文件执行，而是由特定解释器对脚本解释执行，所以只要系统上有相应的解释器就可以做到跨平台运行。Python 脚本比较特殊的是，在直接执行 Python 脚本时是解释执行，但若在该脚本中导入了另一个模块，这个模块会产生.pyc 字节码文件。

应用

以下是一个普通的 Python 脚本文件，实现合并多个 html 为可用文本。

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

""" Html To Text
@author: lizzie
@contract: shengyan1985@gmail.com
@see: ...
@version:0.1
"""
import os
from html2text import *
import chardet
import sys
```

```
14
15 reload(sys)
16 sys.setdefaultencoding('utf8')
17
18 YAHOO_DIR = 'J:\\yahoo_data\\'
19 YAHOO_TXT = YAHOO_DIR+ 'txt\\all.txt'
20
21 def html_to_txt():
22     """将多个 html 文件合并为一个 txt 文件，统一编码为 utf-8 or ascii
23     """
24     ft = open(YAHOO_TXT, 'w')
25     start = 1
26     while 1:
27         filename = YAHOO_DIR+ str(start) + '.html'
28         if os.path.isfile(filename):
29             fp = open(filename, 'r')
30             htmltxt = ''.join(fp.readlines())
31             if not htmltxt or not len(htmltxt):
32                 continue
33             fp.close()
34
35             codedetect = chardet.detect(htmltxt)["encoding"]
                                                    #检测得到修改之前的编码方式
36             print codedetect
37             if not codedetect in ['utf-8', 'ascii']:
38                 htmltxt = unicode(htmltxt, codedetect).encode('utf-8')
39                 codedetect = chardet.detect(htmltxt)["encoding"]
                                                    #检测得到修改之后的编码方式
40                 print 'change', codedetect
41
42             ft.write(html2txt(htmltxt))
43             print 'Success change html to txt %s' % start
44             start += 1
45         else:
46             break
47     ft.close()
48
49 if __name__ == '__main__':
50     html_to_txt()
```

接下来依次介绍 Python 脚本的各个部分。

`#!/usr/bin/python` 这句话表示该脚本文件用哪个解释器来执行，这里是 Python 解释器；

另一种写法是#!/usr/bin/env python。两种写法是有区别的，区别详述请见网址：

http://groups.google.ro/group/python-cn/msg/843a13f6d8be7eab

精巧地址：*http://bit.ly/3TFHPC*

`# -*- coding: utf-8 -*-`指定字符编码方式。有关字符编码方式和字符集的相关知识参见 PCS6 和以下这个链接。

字符编码和字符集介绍：

http://www.ruanyifeng.com/blog/2007/10/ascii_unicode_and_utf-8.html

精巧地址：*http://bit.ly/2BEWnC*

`import os` 表示导入 os 模块。`from html2text import *`、`import chardet`、`import sys` 同样是导入相应模块。这里的 html2txt 实现的是将某个 html 文件去除 html 标签，提取可用信息，转变成纯文本。

定义全局变量。全局变量通常用大写字母来标识。

定义函式。其中:表示函式开始，函式内利用缩进体现块与块之间的不同。#后表示注释。

最后调用函式执行。

执行脚本。在命令行中输入 python pcs-5-1.py 就可以执行。

小结

Python 脚本文件的结构是非常清晰的，很容易掌握。

PCS6　Python 与中文

概述

中文编码总是个难题。不过 Python 可以完美地支持中文，不管你的字符编码是 GB2312，GBK 或 UTF-8。在这里，不准备讲述有关 ASCII/Unicode 编码和中文编码问题，这两部分内容可以分别参考字符编码介绍和汉字编码介绍：

字符编码介绍：*http://www.ruanyifeng.com/blog/2007/10/ascii_unicode_and_utf-8.html*

精巧地址：*http://bit.ly/2BEWnC*

汉字编码介绍：*http://www.css8.cn/css8_document/gb2312.htm*

精巧地址：*http://bit.ly/2Oxpp7*

下面介绍 Python 中的中文编码问题。

应用

Python 中文编码转换，主要有以下 4 个对象来处理字符串转换操作等事项。

```
class str(basestring)
 |  str(object) -> string
 |
 |  Return a nice string representation of the object.
 |  If the argument is a string, the return value is the same object.
......
```

```
class unicode(basestring)
 |  unicode(string [, encoding[, errors]]) -> object
 |
 |  Create a new Unicode object from the given encoded string.
 |  encoding defaults to the current default string encoding.
 |  errors can be 'strict', 'replace' or 'ignore' and defaults to 'strict'.
......
encode(...)
   S.encode([encoding[,errors]]) -> object

   Encodes S using the codec registered for encoding. encoding defaults to
   the default encoding. errors may be given to set a different error handling
   scheme. Default is 'strict' meaning that encoding errors raise a
   UnicodeEncodeError. Other possible values are 'ignore', 'replace' and
   'xmlcharrefreplace' as well as any other name registered with
   codecs.register_error that is able to handle UnicodeEncodeErrors.
decode(...)
   S.decode([encoding[,errors]]) -> object

   Decodes S using the codec registered for encoding. encoding defaults to
   the default encoding. errors may be given to set a different error handling
   scheme. Default is 'strict' meaning that encoding errors raise a
   UnicodeDecodeError. Other possible values are 'ignore' and 'replace' as
   well as any other name registerd with codecs.register_error that is able
   to handle UnicodeDecodeErrors.
```

`str` 表示将某对象转换成字符串表示。

`unicode` 表示将某个字符串按照某个编码方式转换为一个 unicode 对象，unicode 是内部编码，世界上所有字符的统一编码集，即每个字符都有唯一的二进制表示方法，可以在 *http://www.unicode.org/* 上查到某个字符的 unicode 码。

在 Python 中，如果要正确显示中文字符，简单做法就是把字符串先转化为 unicode，然后再由 unicode 对象转化成任意其他系统可以显示的编码。如果你的操作系统是 GNU/Linux 的，只能正确显示 UTF-8，而你要显示的文本是从一个 Windows 系统下过来的 GB2312 编码，那么你要做的就是将 GB2312 转化成 unicode，然后由 unicode 转化成 UTF-8，这样就可以正确显示了。

```
In [26]: f = open('test')

In [27]: line = f.readline()

In [28]: uline = unicode(line, 'gb2312')
```

```
In [29]: uline
Out[29]: u'lines: 0.838643\n'

In [30]: u8line = uline.encode('utf-8')

In [31]: u8line
Out[31]: 'lines: 0.838643\n'
```

`encode` 将对象按照指定编码进行编码。

`decode` 将对象按照指定编码进行解码。

```
In [6]: s = "你好"

In [7]: s.decode("gbk")
Out[7]: u'\u6d63\u72b2\u30bd'

In [8]: s.decode("gbk").encode("utf-8")
Out[8]: '\xe6\xb5\xa3\xe7\x8a\xb2\xe3\x82\xbd'
```

或者

```
In [12]: s = "你好"

In [13]: unicode(s, "gbk")
Out[13]: u'\u6d63\u72b2\u30bd'

In [14]: unicode(s, "gbk").encode("utf-8")
Out[14]: '\xe6\xb5\xa3\xe7\x8a\xb2\xe3\x82\xbd'
```

小结

字符编码是个比较困扰人的问题，因为世界上有太多不同的编码方式，如果全世界统一用一个字符集和一种编码实现方式，那么就不会有这么多问题了，但这是不可能的。有关字符编码的知识还可以在下面的链接中找到更多。

- 谈谈 Unicode 编码：*http://www.pconline.com.cn/pcedu/empolder/gj/other/0505/616631.html*
 精巧地址：*http://bit.ly/17VlhU*
- 字符编码笔记：ASCII，Unicode 和 UTF-8：*http://www.ruanyifeng.com/blog/2007/10/ascii_unicode_and_utf-8.html*

精巧地址：*http://bit.ly/2BEWnC*

- 汉字编码问题：*http://www.css8.cn/css8_document/gb2312.htm*
 精巧地址：*http://bit.ly/2Oxpp7*
- 如何解决 Python 中文编码问题：*http://webclipping.com.cn/2007/12/18/如何解决Python-中文编码问题/*
 精巧地址：*http://bit.ly/vgeZ*

PCS7　Python 编码规范

概述

本文参考自 Python 增进提案仓库 008 号文件的倡议。

原文“PEP 008 《Style Guide for Python Code》”：*http://www.python.org/dev/peps/pep-0008/*

精巧地址：*http://bit.ly/2OQ8Z3*

其中文翻译：*http://wiki.woodpecker.org.cn/moin/PythonCodingRule*

精巧地址：*http://bit.ly/3HURoL*

在这里先简要叙述一下要点。

应用

Python 八荣八耻

以动手实践为荣，以只看不练为耻；

以打印日志为荣，以单步跟踪为耻；

以空格缩进为荣，以制表缩进为耻；

以单元测试为荣，以人工测试为耻；

以模块复用为荣，以复制粘贴为耻;

以多态应用为荣，以分支判断为耻;

以 Pythonic 为荣，以冗余拖沓为耻;

以总结分享为荣，以跪求其解为耻。

这首诗出自于 CPyUG:40912：*http://groups.google.com/group/python-cn/browse_thread/thread/71c74c4e77577365/a4940f98b59bde43*

精巧地址：*http://bit.ly/4jeBor*

它非常简明地指出了学习 Python 的多个注意点。

一致性的建议

愚蠢地使用一致性是无知的妖怪（A Foolish Consistency is the Hobgoblin of Little Minds）。这里的一致性主要是指一个项目内的一致性和一个模块或函式内的一致性，相对于前者而言，后者更为重要。但最重要的是知道何时会不一致。当出现不一致时，运用自己的最佳判断，看看别的例子，然后决定怎样看起来更好。

代码的布局

缩进

建议使用 Emacs 的 Python-mode 默认值：4 个空格一个缩进层次。对于确实古老的代码，若不希望产生混乱，可以继续使用 8 空格的制表符。在 Emacs 的 Python-mode 中会自动发现文件中主要的缩进层次，依此设定缩进参数。如果使用其他的编辑器，如 vim、gedit、ulipad 等，积极建议把 4 个空格作为一个缩进层次。

制表符还是空格

永远不要混用制表符和空格，因为如果混用了，虽然在编辑环境中显示两条语句为同一缩进层次，但因为制表符和空格的不同会导致 Python 解释为两个不同的层次。

最流行的 Python 缩进方式是仅使用空格，其次是仅使用制表符。若一定要混合使用制表符和空格，可以将其转换成仅使用空格，如在 Emacs 中，选中整个缓冲区，按 ESC+X 键去除制表符；或者在调用 Python 命令行解释器时使用-t 选项，可对代码中不合法的混合制表符和空格发出警告，使用-tt 时警告将变成错误，这些选项是被高度推荐的。但是强烈推荐仅使用空格而不是制表符。

行的最大长度

有许多设备被限制为每行 80 字符，窗口也限制为 80 个字符，因此，建议将所有行限制在最大 79 字符（Emacs 准确地将行限制为长 80 字符）。

对顺序摆放的大块文本（文档字符串或注释），推荐将长度限制为 72 字符。折叠长行的首选方法是使用 Pyhon 支持的圆括号、方括号或花括号内的行延续。如果需要，可以在表达式周围增加一对额外的圆括号，但是使用反斜杠看起来会更好，比如下面这个例子：

```
1       class Rectangle(Blob):
2             def __init__(self, width, height,
3                          color='black', emphasis=None, highlight=0):
4                   if width == 0 and height == 0 and \
                    color == 'red' and emphasis == 'strong' or \
                    highlight > 100:
5                         raise ValueError, "sorry, you lose"
6                   if width == 0 and height == 0 and (color == 'red' or
7                                                      emphasis is None):
8                         raise ValueError, "I don't think so"
9                   Blob.__init__(self, width, height,
10                                color, emphasis, highlight)
```

空行

用两行空行分割顶层函式和类的定义，类内方法的定义用单个空行分割。

额外的空行可被用于分割一组相关函式。在一组相关的单句中间可以省略空行。

当空行用于分割方法的定义时，在 class 行和第一个方法定义之间也要有一个空行。

在函式中使用空行时，请谨慎地将它用于表示一个逻辑段落。Python 接受 Control+L（即 ^L）换页符作为空格；Emacs（和一些打印工具）视这个字符为页面分割符，因此在文件中，可以用它们来作为相关片段分页。

导入

通常应该在单独的行中导入，例如：

```
No:  import sys, os
Yes: import sys
       import os
```

但是这样也是可以的：

```
1 from types import StringType, ListType
```

imports 通常被放置在文件的顶部，仅在模块注释和文档字符串之后，在模块的全局变量和常量之前。imports 应该有顺序地成组安放，顺序依次为：标准库的导入、 相关的主包的导入、特定应用的导入。同时建议在每组导入之间放置一个空行。

对于内部包的导入是不推荐使用相对路径的，而对所有导入都要使用包的绝对路径。

从一个包含类的模块中导入类时，通常可以写成这样：

```
1 from MyClass import MyClass
2 from foo.bar.YourClass import YourClass
```

如果这样写导致了本地名字冲突，那么就这样写：

```
1 import MyClass
2 import foo.bar.YourClass
```

即可以使用 MyClass.MyClass 和 foo.bar.YourClass.YourClass 这种写法。

空格

建议不要在以下地方出现空格：

- 紧挨着圆括号、方括号和花括号的地方，如 spam(ham[1], { eggs: 2 })，要始终将它写成 spam(ham[1], {eggs: 2})
- 紧贴在逗号、分号或冒号前的地方，如 if x == 4 : print x , y ; x , y = y , x，要始终将它写成 if x == 4: print x, y; x, y = y, x
- 紧贴着函式调用的参数列表前开式括号（open parenthesis ）的地方，如 spam (1)，要始终将它写成 spam(1)
- 紧贴在索引或切片开始的开式括号前的地方，如 dict ['key'] = list [index]，要始终将它写成 dict['key'] = list[index]

在赋值（或其他）运算符周围，用于和其他并排的一个以上的空格，如：

```
1 x           = 1
2 y           = 2
3 long_variable = 3
```

要始终将它写成

```
1 x = 1
2 y = 2
3 long_variable = 3
```

始终在这些二元运算符两边放置一个空格，如赋值（=）、比较（==, <, >, !=, <>, <=,>=, in, not in, is, is not）、布尔运算（and, or, not）。

在算术运算符周围插入空格，始终保持二元运算符两边的空格一致。

不要在用于指定关键字参数或默认参数值的=号周围使用空格。不要将多条语句写在同一行上，如：

```
No:  if foo == 'blah': do_blah_thing()
Yes: if foo == 'blah':
          do_blah_thing()
No:  do_one(); do_two(); do_three()
Yes: do_one()
     do_two()
     do_three()
```

文档化

为所有公共模块、函式、类和方法编写文档字符串。文档字符串对非公开的方法不是必要的，但你应该有一个描述这个方法做什么的注释。

多行文档字符串结尾的""" 应该单独成行，例如：

```
"""Return a foobang
Optional plotz says to frobnicate the bizbaz first.
"""
```

对单行的文档字符串，结尾的"""在同一行也可以。

版本注记

如果要将 RCS 或 CVS 的杂项包含在你的源文件中，按如下格式操作：

```
1 __version__ = "$Revision: 1.4 $"
```

```
    2 # $Source: E:/cvsroot/python_doc/pep8.txt,v $
```

对于 CVS 的服务器工作标记更应该在代码段中明确它的使用说明，如在文档最开始的版权声明后应加入如下版本标记：

文件：id

版本：$Revision$

这样的标记在提交给配置管理服务器后，会自动适配成为相应的字符串，如：

文件：$Id: ussp.py,v 1.22 2004/07/21 04:47:41 hd Exp $

版本：$Revision: 1.4 $

这些内容应该包含在模块的文档字符串之后，所有代码之前，上下用一个空行分割。

命名约定

以下的命名风格是众所周知的。

- b （单个小写字母）
- B （单个大写字母）
- 小写串 如：getname
- 带下画线的小写字符串，如_getname
- 大写串，如 GETNAME
- 带下画线的大写字符串，如_GETNAME
- CapitalizedWords（首字母大写单词串）
- mixedCase（混合大小写单词串）
- Capitalized_Words_With_Underscores（带下画线的首字母大写单词串）

另外，以下用下画线作前导或结尾的特殊形式是被公认的（这些通常可以和任何习惯一起使用）。

- `_single_leading_underscore`(以一个下画线作前导)：弱的“内部使用(internal use)”标志。例如，“`from M import*`”不会导入以下画线开头的对象。
- `single_trailing_underscore_`（以一个下画线结尾）：用于避免与 Python 关键词的冲突，例如“`Tkinter.Toplevel(master, class_='ClassName')`”。
- `__double_leading_underscore`（双下画线）：从 Python 1.4 起为类私有名。
- `__double_leading_and_trailing_underscore__`：特殊的（magic） 对象或属性，存在于用户控制的(user-controlled)名字空间，例如：`__init__`、`__import__`或`__file__`。

有时它们被用户定义，用于触发某个特殊行为（magic behavior）（例如：运算符重载）；有时被构造器（infrastructure）插入，以便自己使用或为了调试。因此，在未来的版本中，构造器（也可定义为 Python 解释器和标准库）可能打算建立自己的魔法属性列表，用户代码通常应该限制将这种约定作为己用。成为构造器的一部分的用户代码可以在下画线中结合使用短前缀，例如：`__bobo_magic_attr__`。

应避免的名字则有：永远不要用字符 l（小写字母 el（就是读音，下同））、O（大写字母 oh），或者 I（大写字母 eye）作为单字符的变量名。在某些字体中，这些字符不能与数字 1 和 0 分开。当想要使用 l 时，用 L 代替它。

模块名：模块应该是不含下画线的、简短的、小写的名字。因为模块名被映射到文件名，有些文件系统大小写不敏感并且截短长名字，模块名被设为相当短是重要的——这在 Unix 上不是问题，但当代码传到 Mac 或 Windows 上就可能是个问题了。Python 包应该是不含下画线的，简短的，全小写的名字。

类名：几乎没有例外，类名总是使用首字母大写单词串（CapWords）的约定。

异常名：如果模块对所有情况定义了单个异常，它通常被叫做“error”或“Error”。似乎内建的模块使用“error”（例如：os.error），而 Python 模块通常用“Error”（例如：xdrlib.Error）。

全局变量名：一般全部大写字母命名。

函式名：函式名应该为小写，可用下画线风格单词以增加可读性。

方法名和实例变量：这大体上和函式相同，通常使用小写单词，必要时用下画线分隔增加可读性。使用一个前导下画线仅用于不打算作为类的公共接口的内部方法和实例变量。Python 不强制要求这样，它取决于程序员是否遵守这个约定。使用两个前导下画线以表示类私有的名字。Python 将这些名字和类名连接在一起：如果类 Foo 有一个属性名为`__a`，它不能以 `Foo.__a` 访问。通常，双前导下画线应该只用来避免与类（为可以子类化所设计）中的属性发生名字冲突。

继承的设计：始终要确定一个类中的方法和实例变量是否要被公开。通常，永远不要将数据变量公开，除非你实现的本质只是记录。人们总是更喜欢给类提供一个函式的接口作为替换。同样，确定你的属性是否应为私有的。私有与非公有的区别在于： 前者永远不会被用在一个派生类中，而后者可能会。私有属性必须有两个前导下画线，无后置下画线。非公有属性必须有一个前导下画线，无后置下画线。公共属性没有前导和后置下画线，除非它们与保留字冲突，在此情况下，单个后置下画线比前置或混乱地拼写要好，例如：class_优于 klass。

其他建议

要对像 None 之类的单值进行比较，应该永远用：is 或 is not 来做。当你本意是 if x is not None 时，写成 if x 要小心，例如当你测试一个默认为 None 的变量或参数是否被设置为其他值时。这个其他值可能是一个在布尔上下文中为假的值!

基于类的异常总是好过基于字符串的异常。模块和包应该定义它们自己的域内特定的基异常类，基类应该是内建的 Exception 类的子类。还始终包含一个类的文档字符串。例如：

```
1 class MessageError(Exception):
2       """Base class for errors in the email package."""
```

应当使用字符串方法代替字符串模块，除非必须向后兼容 Python 2.0 以前的版本。字符串方法运行总是非常快，而且和 unicode 字符串共用同样的 API（应用程序接口）

在检查前缀或后缀时避免对字符串进行切片。用 startswith()和 endswith()代替，因为它们是明确的并且错误更少。例如：

```
No:  if foo[:3] == 'bar':
Yes: if foo.startswith('bar'):
```

对象类型的比较应该始终用 isinstance()代替直接比较类型。例如：

```
No:  if type(obj) is type(1):
Yes: if isinstance(obj, int):
```

检查一个对象是否是字符串时，紧记它也可能是 unicode 字符串！在 Python 2.3、str 和 unicode 有公共的基类（basestring），所以你可以这样做：

```
1 if isinstance(obj, basestring):
```

对序列（字符串（strings）、列表（lists）、元组（tuples））使用空列表是 false 这个事实，因此 if not seq 或 if seq 比 if len（seq）或 if not len（seq）好。

书写字符串文字时不要有意后置空格。这种后置空格在视觉上是不可辨认的，并且有些编辑器会将它们修整掉。

不要用 == 来比较布尔型的值以确定是 True 或 False。

```
No:  if greeting == True:
Yes: if greeting:
No:  if greeting == True:
Yes: if greeting:
```

语法篇

PCS100	import	200
PCS101	内建数据类型	205
PCS102	For 循环	214
PCS103	缩进	218
PCS104	注释	220
PCS105	对象	222
PCS106	文件对象	225
PCS107	字串格式化	228
PCS108	函式	230
PCS109	系统参数	235
PCS110	逻辑分支	238
PCS111	类	241
PCS112	异常	244
PCS113	交互参数	247
PCS114	FP 初体验	249

PCS100　import

模块及包的使用

概述

Python 语言是通过 import 或 from import 语句来调用模块的。如果程序文件或模块比较多，则会导致整个目录杂乱无章，为了便于管理各个模块，我们把各个模块分门别类存放在不同的文件夹下，并把单独存放模块的文件夹称做包。

应用

模块(modules)

模块（modules）其实就是实现一定具体功能的普通 Python 脚本文件，命名规则是模块名称后加上.py 后缀。主要供其他程序将其引入，以便利用其提供的操作、功能和数据，Python 标准库全部是以模块方式提供的。例如：fibo 模块（fibo.py）是一个实现 Fibonacci 功能的模块。

```
1 # -*- coding: utf-8 -*-
2 # Fibonacci 数列模块
3 # 输出所有小于 n 的 Fibonacci 数
4 def fib(n):
5    a, b = 0, 1
6    if n == 1:
7       print 1
```

```
 8    while b < n:
 9        print b,
10        a, b = b, a+b
11
12 # 返回所有小于 n 的 Fibonacci 数
13 def fib2(n):
14    result = []
15    a, b = 0, 1
16    while b < n:
17        result.append(b)
18        a, b = b, a+b
19    return result
```

在 Python 解释器中，使用 import fibo 语句导入 fibo 模块，使用 fibo.fib(1000)来调用函式，也可以用 fib = fibo.fib 将模块函式赋值到本地函式。

```
>>> import fibo
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
>>> fib = fibo.fib
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
>>> from fibo import fib,fib2
>>> fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

正如上述代码中，不仅可以通过 import 来实现模块中函式的使用，还可以通过 from ...import 方式来导入模块中的函式。

包(package)

包是采用“.”组织模块命名空间方式的，比如模块名称 A.B 是表示 A 包中的模块 B。这种命名空间的组织方式能够避免不同模块命名的冲突。例如：设计一组模块来处理声音文件和声音数据，即如何组织一个包。由于存在多个不同声音格式的文件，需要一个随时能增加新模块的包来处理新增的声音格式。另外还需要对声音进行各种不同处理（例

如混声、加回音、加入平衡、加入人工音效等），所以还需要另写一些模块来作这些处理。如以下组织结构：

```
Sound/                          Top-level package
      __init__.py               Initialize the sound package
      Formats/                  Subpackage for file format conversions
            __init__.py
            wavread.py
            wavwrite.py
            aiffread.py
            aiffwrite.py
            auread.py
            auwrite.py
            ...
      Effects/                  Subpackage for sound effects
            __init__.py
            echo.py
            surround.py
            reverse.py
            ...
      Filters/                  Subpackage for filters
            __init__.py
            equalizer.py
            vocoder.py
            karaoke.py
            ...
```

__init__.py 是必须的，它帮助 Python 将该目录识别为包。在最简单的例子中，__init__.py 是一个空文件。当然也可以让__init__.py 做一些包的初始化动作或是设定一些变量，如__all__变量。

import

直接导入包中的一个模块或导入模块中定义的一个函式（如模块 fibo 的函式 fib）来使用，比如：

```
1 # -*- coding: utf-8 -*-
2 import Sound.Effects.echo
3 # 使用这个模块,必须使用完整的名字来调用
4 Sound.Effects.echo.echofilter(input, output, delay=0.7, atten=4)
5
6 # 另一种替代方法
```

```
 7 from Sound.Effects import echo
 8 # 不同的是，不需要包前缀
 9 echo.echofilter(input, output, delay=0.7, atten=4)
10
11 # 另一种直接导入你需要的函式和变量的方法
12 from Sound.Effects.echo import echofilter
13 # 其使用方法为
14 echofilter(input, output, delay=0.7, atten=4)
```

from ... import

另一种写法 from Sound.Effects import * 会怎么样？ 理想情况下，可能期望它会搜寻整个包目录，然后搜寻所有的模块并且一一导入。但是，在 Mac 及 Windows 平台下，文件的名称大小写不一致，无法保证所有的模块都会被导入。所以唯一的解决方法就是包的作者提供一个明确索引给使用包的人。如果遵守该习惯的话，当使用包的人在导入的时候使用 from Sound.Effects import *，就会查找包中的`__init__.py`中的`__all__`这个 list 变量，该变量就包含所有应该被导入的模块名称。身为包的作者有责任维护更新`__init__.py`。以 `Sounds/Effects/__init__.py` 为例：

```
1 __all__ = ["echo", "surround", "reverse"]
```

表示当 from Sound.Effects import * 时会 import 这三个 module。如果没有定义`__all__`，from Sound.Effects import *不会保证所有的子模块被导入。所以要么通过`__init.py__`，要么显式地 import 以保证子模块被导入。如下所示：

```
1 import Sound.Effects.echo
2 import Sound.Effects.surround
3 from Sound.Effects import *
```

值得注意的是 import *不被鼓励，因为这样会降低程序的可读性，虽然这样会减少一些打字工作。有些模块在设计时故意只让某些特别的名称可以被使用，如使用 from Package import specific_submodule，这样做没有任何不对，但如果你的模块名称和其他名称冲突，就得使用 as 为冲突模块取个别名：

```
1 from Package import specific_submodule as specific_submodule_alias
```

搜寻路径

当你导入 fibo 时，Python 解释器先在当前目录下搜寻 fibo.py 文件，如果没有找到，会依

次在$PYTHONPATH 指示的所有路径中搜寻。$PYTHONPATH 的设定方法与$PATH 是一样的，即多个目录路径的字符串。事实上，模块的搜寻路径是依照 sys.path 变量（多个路径组成 的 list 变量）来的。当 Python 解释器启动时，会将当前目录、$PYTHONPATH，以及按照安装时设定的一些目录加入 sys.path 变量中，所以可以修改这些参数来控制搜寻模块的路径。例如：

```
1 # -*- coding: utf-8 -*-
2 import sys
3 # 将 fibo.py 的路径添加到 sys.path
4 sys.path.append('/home/shengyan/workspace/')
5
6 from pcs import fibo
```

小结

本文简要介绍了模块、包的相关知识，同时介绍了 import、from import 的使用及相关内容，更多的阅读资料可查看下面的资源链接。

- LpyQLearn-6-model：
 http://wiki.woodpecker.org.cn/moin/ObpLovelyPython/LpyQLearn-6-model
 精巧地址：*http://bit.ly/3ZiH8s*
- Importing Python Modules：*http://effbot.org/zone/import-confusion.htm*，这是很适合于初学者阅读的一篇文章。
 精巧地址：*http://bit.ly/2xqCpU*
- Modules 文档：*http://docs.python.org/tut/node8.html*
 精巧地址：*http://bit.ly/px6V6*
- import 语句：*http://docs.python.org/ref/import.html*
 精巧地址：*http://bit.ly/3H9UUC*

PCS101　内建数据类型

概述

Python 提供的基本数据类型是比较丰富的，主要有：布尔类型、整型、浮点型、字符串、列表、元组、集合、字典，等等。下面就依次介绍它们的使用方法。

应用

空(None)

None 表示该值是一个空对象，且对其没有特别的操作，比如没有明确定义返回值的函式就返回 None。

布尔类型(Boolean)

在 Python 中，None、任何数值类型中的 0、空字符串"、空元组()、空列表[]、空字典{}都被当作 False，还有自定义的类型，如果它实现了__nonzero__()或__len__()方法且方法返回 0 或 False，则其实例也被当作 False，其他对象均为 True。

```
>>> bigornot = 1>2
```

```
>>> bigornot
False
>>> if []:
...     print "True"
... else:
...     print "False"
...
False
>>> if '':
...     print "True"
... else:
...     print "False"
...
False
```

整型(Int)

在 Python 内部对整数的处理分为普通整数和长整数，普通整数长度为机器位长，通常都是 32 位。超过这个范围的整数就自动当作长整数处理，而长整数表示的范围就几乎完全没有限制了。

```
>>> a = []
>>> a is False
False
>>> a = 1
>>> type(a)
<type 'int'>
>>> b = 999999999999999999999999
>>> type(b)
<type 'long'>
>>> a
1
>>> b
999999999999999999999999L
```

浮点型(Float)

Python 的浮点数就是数学中的小数，类似 C 语言中的 double。在数值运算中，整数与浮点数运算的结果是浮点数，这就是所谓的“提升规则”，也就是“小”类型会被提升为

“大”类型参与计算，许多语言都是这样的。在 Python 的所有运算中，对应类型所表示范围小的一般会被扩宽到范围大的类型，其依次为：int、long、 float、complex。其中，complex 包括像 list 这样的集合体。

```
>>> a = 0.000000001
>>> type(a)
<type 'float'>
>>> a
1.0000000000000001e-09
```

字符串(String)

Python 字符串既可以用单引号括起来也可以用双引号括起来，甚至还可以用三引号括起来。这样如果字符串里本身包含双引号，就可以用单引号来括，也可以用双引号加转义字符来括：

```
>>> 'My name is "python" '
'My name is "python" '
>>> 'My name is \"python\" '
'My name is "python" '
>>> "My name is \"python\" "
'My name is "python" '
```

同样，如果字符串里本身就包含单引号呢？可以使用双引号，也可以结合使用单引号和转义字符来实现：

```
>>> "My name is 'python' "
"My name is 'python' "
>>> 'My name is \'python\' '
"My name is 'python' "
```

三个引号的字符串就更方便了，中间甚至还可以换行！

```
>>> '''My
... name
... is
... "python"
... !
... '''
'My\nname\nis\n"python"\n!\n'
```

列表(List)

用符号[]表示列表，中间的元素可以是任何类型，用逗号分隔。list 类似于 C 语言中的数组，用于顺序存储结构。

```
>>> test = [1, 2, "yes"]
```

内建函式

1. append(x) 追加到链尾。
2. extend(L) 追加一个列表，等价于+=。
3. insert(i,x) 在位置 i 插入 x，其余元素向后推。如果 i 大于列表的长度，就在最后添加。如果 i 小于 0，就在最开始处添加。
4. remove(x) 删除第一个值为 x 的元素，如果不存在会抛出异常。
5. reverse() 反转序列。
6. pop([i]) 返回并删除位置为 i 的元素，i 默认为最后一个元素(**i 两边的[]表示 i 为可选的，实际不用输入**)。
7. index(x) 返回 x 在列表中第一次出现的位置，不存在则抛出异常。
8. count(x) 返回 x 出现的次数。
9. sort() 排序。
10. len(List) 返回 List 的长度。
11. del list[i] 删除列表 list 中指定的第 i+1 个变量。

```
>>>test = [1,2,"yes"]
>>>test.append(1) # 追加到链尾
>>>test
[1, 2, 'yes', 1]
>>>test.extend([ 'no','maybe']) # 追加一个列表
>>>test
[1, 2, 'yes', 1, 'no', 'maybe']
>>> test.insert(0,'never') # 在位置 0 插入'never'
>>> test
['never', 1, 2, 'yes', 1, 'no', 'maybe']
>>> test.remove('no') # 删除第一个值为"no"的元素,如果不存在会抛出异常
>>> test
```

```
['never', 1, 2, 'yes', 1, 'maybe']
>>> test.reverse() # 反转序列
>>> test
['maybe', 1, 'yes', 2, 1, 'never']
>>> test.pop() # 返回并删除位置为 i 的元素,i 默认为最后一个元素
'never'
>>> test
['maybe', 1, 'yes', 2, 1]
>>> test.index('yes') # 返回第一个值为'yes'的元素,不存在则抛出异常
2
>>> test.count(1) # 返回 1 出现的次数
2
>>>test.sort() # 排序
>>> test
[1, 1, 2, 'maybe', 'yes']
>>> len(test)
5
>>> del test[2] # 删除
>>> test
[1, 1, 'maybe', 'yes']
>>>
```

切片

切片指的是抽取序列的一部分，其形式为：list[start:end:step]。其抽取规则是：一般默认的步长为 1，但也可自定义。在默认步长的情况下，抽取的部分应该是 list 中从 start 直到(end−1)；step=1 时，抽取的部分应该是 list 中从 start 开始，每次加上 step，直到 end 为止。当 start 没有给出时，默认从 list 的第一个元素开始；当 end=−1 时表示 list 的最后一个元素，依此类推。

```
>>> test = ['never', 1, 2, 'yes', 1, 'no', 'maybe']
>>> test[0:3] # 包括 test[0],不包括 test[3]
['never', 1, 2]
>>> test[0:6:2] # 包括 test[0],不包括 test[6],而且步长为 2
['never', 2, 1]
>>> test[:-1] # 包括开始,不包括最后一个
['never', 1, 2, 'yes', 1, 'no']
>>> test[-3:] # 抽取最后 3 个
[1, 'no', 'maybe']
>>>test[::-1] # 倒序排列
['maybe', 'no', 1, 'yes', 2, 1, 'never']
```

列表推导式

直接通过 for 循环生成一个 list，形式如下：[<expr1> for k in L if <expr2>]，表示在列表 L 中，如果 expr2 为真，就循环执行 expr1 语句并产生一个列表，此为列表推导式。

```
>>>freshfruit = ['  banana  ','   loganberry  ']
>>>[weapon.strip() for weapon in freshfruit]  # strip()是去除字符串两端多余
                                              的空格，该句是去除序列中的所有字串两端多余的空格
['banana', 'loganberry']
>>>vec = [2,4,6]
>>>[3*x for x in vec if x>3]  # 大于 3 的元素乘上 3 作为新列表元素
[12, 18]
>>>[(x,x**2) for x in vec] # 循环变量要是一个 sequence,而[x,x**2 for x in vec]
                            是错误的
[(2,4),(4,16),(6,36)]
>>>vec2 = [4,3,-9]
>>>[x*y for x in vec for y in vec2] # vec 与 vec2 元素相乘
[8, 6, -18, 16, 12, -36, 24, 18, -54]
>>>[vec[i]+vec2[i] for i in range(len(vec))] # vec 与 vec2 元素相加
[6, 7, -3]
>>>[str(round(355/113.0,i)) for i in range(1,6)]  # str()是转换类型为可以
                           打印的字符，round(x,n)表示对 x 保留 n 位小数(四舍五入)
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

元组(Tuple)

元组是和列表相似的数据结构，但是它一旦初始化就不能更改，速度比 list 快，同时 tuple 不提供动态内存管理的功能。而且，要理解以下规则：

- tuple 可以用下标返回一个元素或子 tuple；
- 表示只含有一个元素的 tuple 的方法是：(d,)后面有个逗号，用来和单独的变量相区分。

```
>>>t = 1234,5567,'hello' # t=(1234,5567,'hello')的简写
>>>x,y,z = t # 拆分操作可以应用于所有 sequence
>>>x
1234
>>>u = t,(1,2,3)
>>>u
((1234,5567,'hello'),(1,2,3))
>>>empty = () # 空元组
```

```
>>>singleton = 'hi', # 单个元素的元组,注意逗号
>>> singleton
('hi',)
>>> print t[2]
hello
>>> print t[:2]
(1234,5567)
```

通过元组可以很简单地进行数据交换。比如：

```
a = 1
b = 2
a,b = b,a
```

集合(Set)

集合是无序的，不重复的元素集，类似数学中的集合，可进行逻辑运算和算术运算。

```
>>> s = set(['a', 'b', 'c'])  # 集合对象 s
>>> len(s)  # 集合 s 的长度
3
>>> 'a' in s  # 元素'a'在集合 s 中, 返回布尔类型
True
>>> 'd' not in s  # 元素'd'不在集合 s 中, 返回布尔类型
True
>>> t = set(['a', 'b', 'c', 'd'])  # 集合对象 t
>>> s.issubset(t)  # s 是否是 t 的子集, 等价于 s <= t
True
>>> s.issuperset(t)  # s 是否是 t 的超集, 等价于 s >= t
False
>>> s.union(t)  # 集合的并, 等价于 s | t
set(['a', 'c', 'b', 'd'])
>>> s | t
set(['a', 'c', 'b', 'd'])
>>> s.intersection(t)  # 集合的交, 等价于 s & t
set(['a', 'c', 'b'])
>>> s & t
set(['a', 'c', 'b'])
>>> s.difference(t)  # 集合的差, 等价于 s - t
set([])
>>> s - t
set([])
>>> t.difference(s)
```

```
set(['d'])
>>> t - s
set(['d'])
>>> s.symmetric_difference(t)  # 集合的异或, 等价于 s ^ t
set(['d'])
>>> s ^ t
set(['d'])
```

字典(Dict)

字典是一种无序存储结构，包括关键字（key）和关键字对应的值（value）。 字典的格式为：dictionary = {key:value}。关键字为不可变类型，如字符串、整数、只包含不可变对象的元组。列表等不能作为关键字。如果列表中存在关键字对，可以用 dict()直接构造字典，而这样的列表对通常是由列表推导式生成的。

```
>>> tel = {'jack':4098, 'sape':4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'jack': 4098, 'guido': 4127}
>>> tel['jack'] # 如果 jack 不存在,会抛出 KeyError
4098
>>> tel.get("zsp", 5000) # 如果"zsp"为 tel 的键则返回其值,否则返回 5000
5000
>>> del tel['sape'] # 删除键'sape'和其对应的值
>>> tel.keys()
['jack', 'guido']
>>> "jack" in tel # 判断"jack"是否 tel 的键
True
>>> "zsp" not in tel
True
>>> for k,v in tel.iteritems():  # 同理 tel.iterkeys()为键的迭代
                                   器,tel.itervalues()为值的迭代器
...     print k,v
...
jack 4098
guido 4127
>>> tel.copy() # 复制一份 tel
{'jack': 4098, 'guido': 4127}
>>> tel.fromkeys([1, 2], 0)
{1: 0, 2: 0}
```

```
>>> tel.popitem() # 弹出一项
('jack', 4098)
```

小结

Python 中的数据类型比较丰富，主要有上述这些。还有很多关于数据类型的知识在这里没有描述，可以进一步参考以下网址：

- 内置数据类型：*http://wiki.woodpecker.org.cn/moin/ObpLovelyPython/LpyQLearn-2-data*
 精巧地址：*http://bit.ly/1FsXZf*
- Python 绝对简明手册：*http://wiki.woodpecker.org.cn/moin/PyAbsolutelyZipManual*
 精巧地址：*http://bit.ly/2EEz6I*

PCS102　For 循环

概述

For 语句是主要的流程控制语句之一，其基本的用法和其他语言类似，特殊之处在于 Python 的 for 语句具有两种不同的风格形式，下面将会详细介绍。

应用

首先举个最简单的例子：

```
for item in [1, 2, 3]:
    print item
```

上面的代码其实就是遍历列表[1, 2, 3]，将其中每一个元素都打印出来。所有的循环语句中都可以使用这么两条语句：break 和 continue。break 表示要退出循环，continue 是说直接进入到下一轮的循环中去：

```
>>> for item in range(10):
...     if item<5:          # 遇到比 5 小的
...         continue        # 进入下一轮循环
...     else:               # 否则遇到的是大于等于 5 的
...         print item      # 输出
...         break           # 并直接退出循环
...
5
```

上面这个例子中的循环体也可以这么写：

```
if item>=5:
    print item
    break
```

for 语句后面可以跟 else 子句，在这里 else 子句的含义是：如果 for 一直循环到了末尾，最后正常退出循环，那么随后就会执行它的 else 子句，否则由于 break 语句或异常等原因退出循环的，则不会执行 else 子句。其实在程序中常常会遇到这样的场景：对某个列表中每一个元素执行某个操作，如果成功执行则马上 break 跳出循环，如果遍历整个列表，发现没有一个元素满足要求，也就是意味着遍历失败，那么处理失败情况的语句就可以放在 else 子句中了，比如：

```
>>> for item in range(10):
...     if item<10:
...         continue
...     else:
...         print item
...         break
... else:
...     print '没有大于等于 10 的数字'
...
没有大于等于 10 的数字
```

如果熟悉 C 语言的话，便会看出 Python 的 for 和 C 的 for 之不同，除了表面上的语法差异，他们的含义更是大相径庭。要类比 Python 这种形式的 for 语句，可以想象某些语言的 for in 或者 foreach 之类的语法。

```
/* c 语言 */
for(int i=0; i<count; i++){
    ...
}

# python
for item in a_iterable:
    ...
```

简单地说，C 语言形式的 for 语句的工作原理是这样的：取第 1 个、第 2 个、第 3 个，一直取到最后一个。而迭代器呢，就是对迭代器取下一个、取下一个、取下一个，一直取到迭代器自己喊停为止。实际上 Python 的 for 语句是同时支持这两种风格的。先来剖析一下 Python 的 for 语句：

```
for item in obj:
    ...
```

如果 obj 对象实现了__iter__方法，就是说它是个迭代器，那这就是迭代器风格的 for 语句，而上面这段代码也就等价于：

```
iterator = iter(obj) # 获取迭代器。
# iter(obj) 等价于 obj.__iter__()
try:
    item = iterator.next() # 取下一个
    ...
    item = iterator.next() # 取下一个
    ...
except StopIteration:     # 迭代器喊停
    pass
```

如果上面的 obj 对象实现了__getitem__方法，也就是说它支持索引操作，这就成了 C 语言风格的那种迭代器，这段代码便等价于：

```
try:
  item = obj[0]     # 取第 0 个
  # obj[0] 等价于 obj.__getitem__(0)
  ...
  item = obj[1]     # 取第 1 个
  ...
except IndexError:     # 取到最后一个
    pass
```

最后再测试一下：

```
>>> class Indexable(object):
...     def __getitem__(self, i):   # 定义__getitem__, 如果 i 大于 10，
                                      就停止迭代
...         if i>10:
...             raise StopIteration()
...         print 'get object %d'%i
...
>>> class Iterable(object):
...     def __init__(self):
...         self.counter = 0
...     def __iter__(self):
...         return self
...     def next(self):
...         if self.counter>10: # 如果计数器大于 10，就停止迭代
...             raise StopIteration()
...         print 'get next, current is %d'%self.counter
...         self.counter += 1 # 计数器增 1
...
```

```
>>> container = Indexable()
>>> for i in container:pass
...
get object 0
get object 1
get object 2
get object 3
get object 4
get object 5
get object 6
get object 7
get object 8
get object 9
get object 10
>>> container = Iterable()
>>> for i in container:pass
...
get next, current is 0
get next, current is 1
get next, current is 2
get next, current is 3
get next, current is 4
get next, current is 5
get next, current is 6
get next, current is 7
get next, current is 8
get next, current is 9
get next, current is 10
```

小结

For 循环是比较重要的控制流语句，本文介绍了 for 语句的基本使用。

- For 循环及相关控制流：
 http://docs.python.org/tut/node6.html#SECTION006200000000000000000
 精巧地址：*http://bit.ly/3it1MI*
- The for statement：*http://docs.python.org/ref/for.html*
 精巧地址：*http://bit.ly/4sKUbJ*

PCS103　缩进

概述

缩进可谓是 Python 的一大特色，估计再没有其他语言在使用缩进作为它语法结构的一部分了。

应用

```
In [5]: f = open('words')

In [6]: result = dict()

In [7]:

In [8]: for line in f.readlines():
   ...:        line = line.strip()
   ...:    tmplist = line.split()
   ...:    for w in tmplist:
   ...:           if result.has_key(w):
   ...:               result[w] += 1
   ...:       else:
   ...:               result[w] = 1
   ...:
```

```
In [9]: for k, v in result.items():
   ...:       print k, '-->', v
```

Python 语言利用缩进来区分不同的块的。不同的块具有不同的作用域，所以在 Python 中，缩进的功能相当于 C 或 Java 中的{}。在上述例子中，三个 for 语句各有自己的块作用域，它们很清晰地利用每行代码的缩进来区分自己属于哪个块。同一个块中具有一致的代码缩进格式，因此 Python 代码文件的结构看起来非常清晰。若本应属于同一块内的代码没有相同的缩进，那么，Python 解释器会认为它们属于不同的语义块，因而会引起异常。

```
In [10]: f = open('words')

In [11]: result = dict()

In [12]: for line in f.readlines():
   ...:     line = line.strip()
   ...:     tmplist = line.split()
   ...:     for w in tmplist:
   ...:         if result.has_key(w):
   ...:             result[w] += 1
   ...:         else:
   ...:         result[w] = 1
------------------------------------------------------------
IndentationError: unindent does not match any outer indentation level
(<ipython console>, line 8)
```

使用缩进有个要注意的地方，就是不要混用制表符和空格，因为虽然表面上看起来是同一层次，但 python 解释器还是会认为是不同的代码缩进。最好仅仅用空格，并且用 4 个空格作为一个缩进层次。

小结

在编写 Python 程序的时候，一定要注意缩进的使用，因为它是用于区分各个语句块的。缩进很容易被熟悉，在学习 Python 的过程中，应该逐渐养成熟练使用缩进的习惯，因为这不仅是语法需要，更能够让我们形成统一、良好的编程风格。

- 代码缩进：
 http://www.woodpecker.org.cn/diveintopy_5_4/getting_to_know_python/indenting_code.html
 精巧地址：*http://bit.ly/12Ys4p*

PCS104　注释

概述

Python 中的注释是以#打头的，用于表明对应代码的语义和功能，以便其他程序员或是自己能更快理解代码的含义。

应用

注释块

注释块通常跟随着一些（或者全部）代码并和这些代码有着相同的缩进层次。注释块中每行以#和一个空格开始（除非他是注释内的缩进文本）。注释块内的段落以仅含单个#的行分割。注释块上下方最好有一空行包围（或上方两行下方一行，这是对一个新函式定义段的注释）。

行内注释

行内注释是和语句在同一行的注释。行内注释应该谨慎使用。它应该至少用两个空格和语句分开。应该以#和单个空格开始，如：

```
1    x = x+1                # x 增 1
```

如果语意很明了，那么行内注释是不必要的，应该被去掉，所以不要这样写。但是有时，这样写是有益的：

```
1    x = x+1                # 边界上的补偿调整
```

多行注释

有时须注释掉多行代码以便调试，这可以使用字符串来实现，即采用三个双引号（"""）将注释内容括起来。同时多行注释时也得注意缩进问题，否则会出错。如

```
 1 # -*- coding: utf-8 -*-
 2 if something is None:
 3    print 'None 值'
 4 else:
 5    # 处理一些事情
 6    print something
 7    """x = 1
 8    y = []
 9    print '这里是个多行注释'
10    """
```

小结

更多有关注释的内容可以参考以下内容。

- Python 编码规范-注释：
 http://wiki.woodpecker.org.cn/moin/PythonCodingRule#head-7d7bdfbe6299504f91653381d6d0b6a390ae8699
 精巧地址：*http://bit.ly/2Cf85F*

PCS105　对象

概述

Python 中的一切东西都是对象，即，一切皆对象！

应用

对象定义

Python 把任何数据抽象为对象。每个对象都有唯一的 id、特定的类型和值。

- 每一个对象都有一个唯一的身份标识自己，任何对象的身份都可以使用内建函式 id()来获得。这个值可以被认为是该对象的内存地址。
- 对象的类型决定了该对象可以保存什么类型的值，可以进行什么样的操作，以及遵循什么样的规则。可以使用内建函式 type()查看 Python 对象的类型。注意，在 Python 中类型也是对象，所以 type()返回的是对象而不是简单的字符串。
- 值则是对象的数据内容。

在创建一个对象后，其 id 不能改变，类型一般情况下不能改变，但对于某些特殊类是可以改变的，而值是可有可无并且也能动态改变的。

Python 有一系列的基本数据类型，这些基本数据类型同样也是对象。另外，也可以创建自定义类型，如下定义了两个类，分别叫 A 和 B：

```
In [32]: class A(object):  # 定义类 A
   ....:    pass
   ....:

In [33]: class B(object):  # 定义类 B
   ....:    pass
   ....:

In [34]: a = A()  # 类 A 的对象 a

In [35]: b = B()  # 类 B 的对象 b

In [36]: type(a)  # 查看对象 a 的类型为类 A
Out[36]: <class '__main__.A'>

In [37]: type(b)  # 查看对象 b 的类型为类 B
Out[37]: <class '__main__.B'>
```

a 和 b 是两个属于不同类的对象。类 A 和 B 继承了 object。Python 中的 object 是最基本的类。

```
class object
 |  The most base type
```

对象属性

不同的 Python 对象具有不同的属性和方法，可以使用点号. 来访问属性。比如定义一个类 Concept：

```
 1 # -*- coding: utf-8 -*-
 2
 3 class Concept(object):
 4        conceptNUM = 0   # 定义类变量
 5
 6        def __init__(self, extent=set(), intent=set()):
 7               Concept.conceptNUM += 1
 8               self.conceptID = Concept.conceptNUM  # 以下 3 个变量是该类的
                                                      数据成员
 9               self.extent = extent
10               self.intent = intent
11
12        def addIntent(self, intentSet): # 以下 2 个自定义方法
13               self.intent = self.intent.union(intentSet)
14
```

```
15        def addExtent(self, extentSet):
16              self.extent = self.extent.union(extentSet)
17
18        def __str__(self):
19              return 'Concept %d: (%s), (%s)\n' % (self.conceptID,
                self.extent, self.intent)
20
21 if __name__ == '__main__':
22    cpt = Concept()
23    print 'Create: ', cpt
24    cpt.addIntent(set([1, 2, 3]))
25    print 'Add Intent: ', cpt
26    cpt.addExtent(set(['a', 'b']))
27    print 'Add Extent: ', cpt
28    print cpt.conceptID
```

可以看到在定义了类 Concept 之后，在主程序中创建了一个对象 cpt，之后使用.来调用其方法和属性。其运行结果为：

```
~$ python pcs-105-1.py
Create:  Concept 1: (set([])), (set([]))
Add Intent:  Concept 1: (set([])), (set([1, 2, 3]))
Add Extent:  Concept 1: (set(['a', 'b'])), (set([1, 2, 3]))

1
```

这里涉及 Python 中类的相关知识，在 PCS111“类”中有详细描述，在此略过。

小结

对于 Python 对象，读者在学习过程中，一定要时刻记住**一切类型皆是对象**，如 int 等基本数据类型也是对象，读者可以查看《Python 源码剖析》来进一步了解 Python 中的对象机制。同时，在 Python 世界中，一切循环是迭代，一切对象是字典，一切命名是引用。

- 关于“一切循环是迭代，一切对象是字典，一切命名是引用”的概念，可访问以下网站：*http://wiki.woodpecker.org.cn/moin/ZoomQuiet/2008-07-01*
 精巧地址：*http://bit.ly/1MfiNV*
- Python Objects：*http://www.effbot.org/zone/python-objects.htm*
 精巧地址：*http://bit.ly/zpdxd*
- Objects, values and types：*http://www.python.org/doc/current/ref/objects.html*
 精巧地址：*http://bit.ly/3v2u3I*

PCS106　文件对象

概述

对文件的操作主要是对文件的读取和写入，当然还有其他的操作，可以通过 help("file") 查看。

应用

写文件

首先看个写文件的例子，它实现了每隔 2 秒钟向 test 中写入一个随机数的功能。

```
 1 # -*- coding: utf-8 -*-
 2 #实现每隔 2 秒钟向 test 中写入一个随机数。
 3 import random
 4 import time
 5
 6 def write_f():
 7     file = open("test", "a")
 8     while 1:
 9         afloat = random.random()
10         print afloat
11         file.write('%s\n' % afloat)
```

```
        file.flush()
        time.sleep(2)
    file.close()

if __name__ == '__main__':
    write_f()
```

`write(str)`：返回为 None，将一字符串 str 写入文件中。

`flush()`：返回 None，把 I/O 缓冲区的内容写入磁盘文件中，这才是真正写入文件。

`close()`：返回 None 或一个整数，关闭文件。

读文件

和上述例子对应的是每秒钟读取 test 中的内容。

```
# -*- coding: utf-8 -*-
#实现每隔 1 秒钟读取 test 中的内容。
import time

def read_f():
    file = open("test", "r")
    while 1:
        content = file.readline()
        if not content:
            time.sleep(1)
        else:
            print content
    file.close()

if __name__ == '__main__':
    read_f()
```

`readline()`：返回下一行的内容，作为字符串，若当前文件指针已为行末，则返回 None。

文件打开方式

在上述两个例子中，使用 `a` 方式打开文件，使得其内容不断更新。使用 `r` 方式则为只读方式，只能读取文件内容。下面将简要介绍相关知识。

- r：以只读方式打开已存在的文件，若文件不存在则抛出异常。此方式是默认方式。

- w：以可写方式打开存在或者不存在的文件。若文件不存在则先新建该文件，若文件存在则覆盖该文件。
- a：用于追加，对*ix 系统而言，所有的内容都将追加到文件末尾而不管探针的当前位置如何。
- b：以二进制方式打开。打开一个二进制文件必须用该模式。增加 b 模式是用来兼容系统对二进制和文本文件处理的不同之处的。
- r+和 a+：以更新方式打开文件（注意 w+覆盖文件）。

小结

对文件的操作是经常要用到的，主要是对文件的读取、写入操作。还有很多的文件操作函式，在这略过，读者可以通过 help(file)查看。

- File Objects：*http://docs.python.org/lib/bltin-file-objects.html*
 精巧地址：*http://bit.ly/3rh8kf*
- 与文件对象共事：
 http://www.woodpecker.org.cn/diveintopython/file_handling/file_objects.html
 精巧地址： *http://bit.ly/1e41SW*

PCS107　字串格式化

概述

字串格式化，就是将字符串按照一定的格式输出。

应用

在 Python 中，使用%作为格式化操作符。它的使用方式为：格式字串 % 值表，非常类似于 C 语言中的 printf()。如：

```
In [1]: a = 1

In [2]: b = 0.999999

In [3]: c = 'somestring'

In [4]: d = (a, b, c)

In [5]: print 'a = %d b = %f c = %s d = %s' % (a, b, c, d)
a = 1 b = 0.999999 c = somestring d = (1, 0.99999899999999997, 'somestring')
```

从这个例子中可以看到，最后的输出格式按照前面的格式化字符串的规定，把后面的值表按照前面的格式化字符串来组织的。后面的值表一般为一元组，但若只有一个值的话，就不必使用元组，直接写这个值即可。另外，也可以使用字典，比如：

```
In [6]: c='你好'
In [7]: d=(a,b,c)
```

```
In [8]: print '%(a)d %(b)f %(c)s %(d)s' % {'a':a, 'b':b, 'c':c, 'd':d}
1 0.999999 你好 (1, 0.99999899999999997, '\xe4\xbd\xa0\xe5\xa5\xbd')
```

使用“%(字典 key)格式化选项”这种方式，即可按照格式输出后面字典中与 key 对应的值。

下面列出的是一些常用的格式化选项：

- d 表示一有符号十进制数。
- i 也表示一有符号十进制数。
- o 表示无符号八进制数。
- u 表示无符号十进制数。
- x 表示无符号十六进制数（小写）。
- X 表示无符号十六进制数（大写）。
- e 表示指数形式的浮点数（小写）。
- E 表示指数形式的浮点数（大写）。
- f 表示浮点数。
- F 也表示浮点数。
- g 表示浮点数（根据值的大小采用%e 或%f）。
- G 表示浮点数（根据值的大小采用%E 或%F）。
- c 表示单个字符（允许是整型和单个字符）。
- r 表示字符串（使用对象的 repr()将其转换为字符串形式）。
- s 表示字符串（使用对象的 str()将其转换为字符串形式）。
- % 表示字符%。

小结

字串格式化，也就是将字符串按照一定的格式输出，应该不难掌握。更多有关字串格式化的资料可参见以下资料。

- String Formatting Operations：*http://docs.python.org/lib/typesseq-strings.html*
 精巧地址：*http://bit.ly/2TH7cE*
- 格式化字符串：
 http://www.woodpecker.org.cn/diveintopython/native_data_types/formatting_strings.html
 精巧地址：*http://bit.ly/3CQSYk*
- 字符串的格式化：*http://blog.csdn.net/magicbreaker/archive/2008/06/09/2525722.aspx*
 精巧地址：*http://bit.ly/4AVfWm*

PCS108　函式

概述

函式是完成特定功能的一些语句块，可以作为一个独立单位使用，若再给它取一个名字，就可以通过这个名字在程序的不同地方多次调用，这样就不须要重复地编写这些语句了。在 Python 语言中，函式也是对象。

应用

函式定义

在 Python 中，通过 `def` 关键字定义函式。def 关键字后跟一个函式名，然后跟一对圆括号，圆括号之中可以包括一些变量名，该行以冒号结尾。接下来，是一块语句，称为函式体。例如：

```
In [1]: def add(a, b):
   ...:     """返回 a 和 b 的和"""
   ...:     return a+b
   ...:
```

这样就定义了函式 add，接受两个参数：a 和 b，接下来是文档字符串，加入一些关于该函式功能的描述，函式体只有一个 return 语句返回结果，通过函式名来调用它：

```
In [3]: add(1, 2)
Out[3]: 3
```

函式参数

参数可以默认，即当调用时没有指定某个参数值，就可用该参数的默认值。

```
In [4]: def add(a, b=1):
   ...:     """返回 a 和 b 的和"""
   ...:     return a+b
   ...:

In [6]: add(5)
Out[6]: 6
```

在 Python 语言中，**类似于参数 b** 的称为关键字参数，而**类似于参数 a** 的称为位置参数，因为它是通过所处的位置来进行参数匹配的。

```
In [6]: add(5)
Out[6]: 6

In [7]: add(5, 2)
Out[7]: 7

In [8]: add(5, b=3)
Out[8]: 8
```

传递关键字参数时无须考虑顺序，而位置参数，则在要传参数的时候须按照参数定义的位置一一对应。也可以以关键字参数的形式传递位置参数，还可以以位置参数的形式传递关键字参数（不推荐，因为这样很容易导致混乱），唯一的规则就是关键字参数一定要在位置参数的右边，并且在指定参数时必须保证其右边的参数值已经指定，否则会引起歧义。

传递参数时，可以使用*和**。*的作用是把序列 args 中的每个元素当作位置参数传进去。**的作用则是把字典 args 变成关键字参数传递。

```
In [9]: def add(*args):
   ...:     print args
   ...:

In [10]: add(1, 2, 3)
(1, 2, 3)
```

```
In [11]: def add(**args):
   ...:     print args
   ...:

In [12]: add(a=1, b=2, c=3)
{'a': 1, 'c': 3, 'b': 2}
```

普通的参数定义和传递方式与 * 可以和平共处，但 * 必须放在所有位置参数的最后，而**必须放在所有关键字参数的最后，否则就要产生歧义。

```
In [13]: def add(a=1,**arg):
   ....:     print a
   ....:     print arg

In [14]: add(a=1,b=2,c=3)
1
{'c': 3, 'b': 2}
```

函式返回

每一个函式都有返回值，即 return 后面的值。若函式中没有 return，则该函式返回为 None。

lambda 函式

lambda 函式是匿名函式，用来定义没有名字的函式对象。在 Python 中，lambda 只能包含表达式！

```
lambda arg1, arg2 ... : expression
```

lambda 关键字后就是逗号分隔的形参列表，冒号后面是一个表达式，表达式求值的结果为 lambda 的返回值。虽然 lambda 的滥用会严重影响代码可读性，不过在适当的时候使用一下 lambda 来减少键盘的敲击还是有其实际意义的，比如做排序的时候，使用 data.sort(key=lambda o:o.year)显然比

```
def get_year(o):
   return o.year
func=get_year(o)
data.sort(key=func)
```

要方便许多！

闭包(closure)

闭包其实就是通常所说的函式嵌套。在嵌套函式的内部函式对象本身包含了外部函式对象的名称空间。

```
In [15]: def Outter(n):
   ....:     def inner(m):
   ....:         return n+m
   ....:     return inner
   ....:

In [16]: Outter(1)(2)
Out[16]: 3

In [17]: Outter(2)(2)
Out[17]: 4
```

装饰器(decorator)

```
@log
def test(a, b):
    pass
```

其中 log 是接受一个函式作为参数同时返回一个新函式的函式，其作用是在调用 test 的前后分别输出 enter test 和 exit test，使用符号@来应用这个装饰器。

```
def log(func):
    def wrapper(*args, **kw):
        print 'enter', func.__name__
        func(*args, **kw)
        print 'exit', func.__name__
    wrapper.__name__ = func.__name__
    wrapper.__globals__ = func.__globals__
    return wrapper
```

log 函式定义另一个叫 wrapper 的嵌套函式，它把所有接受的参数简单地全部传给 func，并在调用前后输出一些信息。最后在对 wrapper 的一些属性进行偷梁换柱之后，将它返回，于是这个 wrapper 就变成了一个被包装过的如假包换的 func 了。

生成器(generator)

```
>>> def number_generator():
...     i = 0
...     while True:
...         yield i
...         i += 1
...
>>> for item in number_generator():
...     print item
...
0
1
2
# 省略后面输出的无穷个数字 Ctrl+c 停止
```

Python 的生成器可以有两种讲法，简单讲，它就是方便地实现迭代器的方法，就像上面代码使用的那样。复杂来讲呢，Python 的生成器其实正是一个神秘的 continuation，实际上大部分时候我们都只需要把 yield 当成是快速实现迭代器的工具来用就行了。神秘的 continuation 已经超出了本书的范畴。

小结

和许多语言不同，在 Python 中，函式是一等公民，函式也是对象，这也是在 Python 中进行编程的基础，理解这一点非常重要。越深入学习 Python，就越能感受到函式在 Python 中的重要地位。

PCS109　系统参数

概述

系统参数，即经常说的函式的形参。在定义函式时，若此函式带有参数，就把这些定义的参数叫做形式参数，当调用这个函式时传入形参的那些变量叫做实际参数。

应用

定义参数

Python 中函式参数通过赋值来进行。在 Python 中函式参数的定义主要有四种方式：

- `F(arg1,arg2,...)`是最常见的定义方式，一个函式可以定义任意个参数，每个参数间用逗号分割，用这种方式定义的参数在调用的时候也必须在函式名后的小括号里提供个数相等的值（实际参数），而且顺序必须相同，也就是说在这种调用方式中，形参和实参的个数必须一致，而且必须一一对应，即第一个形参必须对应第一个实参。例如：

```
1 def a(x,y):
2    print x,y
```

调用该函式，a(1,2)则 x 取 1，y 取 2，形参与实参相对应，如果是 a(1)或者 a(1,2,3)则会报错。

- F(arg1,arg2=value2,...,argN = valueN)这种方式就是第一种的改进版，它提供了默认值。例如：

```
1 def a(x,y=3):
2    print x,y
```

调用该函式，a(1,2)同样还是 x 取 1，y 取 2，但是如果是 a(1)，则不会报错了，这个时候 x 还是 1，y 则为默认的 3。上面这两种方式，还可以更换参数位置，比如 a(y=8,x=3)用这种形式也是可以的。

- 上面两个方式是有多少个形参，就传进去多少个实参，但有时候会不确定有多少个参数，此时这种 F(*arg1)方式则比较有用。它以一个*加上形参名的方式来表示这个函式的实参个数不定，可能为 0 个也可能为 n 个。需要注意的一点是，不管有多少个，在函式内部都被存放在以形参名为标识符的 tuple 中。

```
>>> def a(*x):
...     if len(x)==0:
...         print 'None'
...     else:
...         print x
...
>>> a(1)
(1,)
>>> a()
None
>>> a(1, 2, 3)
(1, 2, 3)
>>> a(x=1, y=2, z=3)#但是若这样传递参数，会报类型错误
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: a() got an unexpected keyword argument 'x'
```

- F(**arg1)形参名前加 2 个 * 表示参数在函式内部将被存放在以形式名为标识符的 dictionary 中，这时调用函式的方法则需要采用 arg1=value1,arg2=value2 这样的形式了。

```
>>> def a(**x):
...     if len(x)==0:
...         print 'None'
...     else:
...         print x
...
>>> a()
None
>>> a(x=1, y=2)
{'y': 2, 'x': 1}
```

```
>>> a(1, 2)    #使用这种方式会报错
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: a() takes exactly 0 arguments (2 given)
```

参数解析

函式参数在调用过程中是按照上述四种方法的优先级依次降低的，也就是先把方式 1 中的 arg 解析，然后解析方式 2 中的 arg=value，再解析方式 3，即把多出来的 arg 这种形式的实参组成 tuple 传进去，最后也就是方式 4，把剩下的 key=value 这种形式的实参组成一个 dictionary 传给带 2 个星号的形参。

```
>>> def test(x, y=1, *a, **b):
...     print x, y, a, b
...
>>> test(1)
1 1 () {}
>>> test(1, 2)
1 2 () {}
>>> test(1, 2, 3, 4)
1 2 (3, 4) {}
>>> test(x=1, y=2)
1 2 () {}
>>> test(1, a=2)
1 1 () {'a': 2}
>>> test(1, 2, 3, a=4)
1 2 (3,) {'a': 4}
>>> test(1, 2, 3, y=4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: test() got multiple values for keyword argument 'y'
```

参数若不能被正确解析，程序会抛出类型错误异常。

小结

函式传递参数时使用*和**分别代表传递的是 tuple 和 dictionary，并且参数是按照一定顺序解析的。

PCS110　逻辑分支

概述

这里讲的逻辑分支主要是指 if ... else ...语句，if ... else ...也是比较重要的流程控制语句之一，使用比较简单。基本语法和其他语言的 if 语句很类似，因为 Python 中没有提供 switch 语句和三元运算符，但可以通过 if 语句来实现。

应用

最基本的用法

以下一段代码使用 if 分支来实现比较两个数大小的功能：

```
 1 # -*- coding: utf-8 -*-
 2
 3 a = int(raw_input("a = "))          #从键盘输入一个 int 类型的变量
 4 b = int(raw_input("b = "))
 5 if a == b:
 6    print 'a 等于 b'
 7 elif a < b:
 8    print 'a 小于 b'
 9 elif a > b:
10    print 'a 大于 b'
```

```
11 else:
12    print '在本宇宙中这是不可能滴！'
```

elif 就是 else if 的缩写。

使用 if 代替 switch

Python 没有 switch 语句，不过 if elif else 组合已经足以应付大部分应用 switch 的情况。而对于另外的一种 switch 应用场景，Python 还有更加优雅的方式来处理：

```
# 伪码 根据输入的不同参数选择程序的不同行为
switch( sys.argv[1] ):
   case '-e':
      walk_cd(); break;
   case '-d':
      search_cd(); break;
   ...
   default:
      raise CommandException("Unknown Commend: " + sys.argv[1])

# 使用 if 替代
if sys.argv[1]=='-e':
   walk_cd()
elif sys.argv[1]=='-d':
   search_cd()
...
else:
   raise CommandException("Unknown Command: " + sys.argv[1])

# 更好的做法
commands = {
   '-e'   : walk_cd,
   '-d'   : search_cd,
}
try:
   commands[ sys.argv[1] ]()
except KeyError:
   raise CommandException("Unknown Command: " + sys.argv[1])
```

最后一种方式不管从可读性（这是显然的）、性能（哈希表 vs 普通查找）上都更好。另外最后一种做法将参数与行为的映射完全独立出来了，一来修改起来极其方便，到时候也很容易将它们分离到配置文件中去。

三元运算符

三元运算符类似于 C 语言中的条件表达式，在 Python 中可以用下述代码实现：

```
>>> def get_length(s):
...     # 等价于:
...     # if(s!=None)
...     #     return len(s)
...     # else:
...     #     return len('None')
...     return len(s) if s!=None else len('None')
...
>>> get_length(None)
4
```

使用 and 和 or 技巧可以将上述代码写成：

```
>>> def get_length(s):
...     return s!=None and len(s) or len('None')
...
>>> get_length(None)
4
```

不过，第一种写法看起来更易读些。

小结

if ... else ...语句是经常用的语法。上面只介绍了其最基本的使用方法，在下面的链接中可以找到更多的资料。

- 过程控制：*http://wiki.woodpecker.org.cn/moin/ObpLovelyPython/LpyQLearn-3-process*
 精巧地址：*http://bit.ly/3RsfJm*
- Python 绝对简明手册：*http://wiki.woodpecker.org.cn/moin/PyAbsolutelyZipManual*
 精巧地址：*http://bit.ly/2EEz6I*

PCS111　类

概述

在 Python 中可以通过关键字 class 定义一个类，后面加上类名称，在类中可以定义类成员和类方法。自定义的类也可以继承现有的类，Python 中类的继承分单继承和多继承。

应用

类定义

自定义的类是基于以下形式的：

```
In [1]: class A(object):
   ...:     """This is Class A"""
   ...:     def __init__(self, a):
   ...:         self.a = a
   ...:

In [2]: class B(object):
   ...:     """This is Class B"""
   ...:     def __init__(self, b):
   ...:         self.b = b
```

```
In [3]: print B.__dict__
{'__dict__': <attribute '__dict__' of 'B' objects>, '__module__': '__main__',
'__weakref__': <attribute '__weakref__' of 'B' objects>, '__doc__': 'This
is Class B', '__init__': <function __init__ at 0x842ed14>}
```

在这个小例子中，关键字 class 定义了两个类，A 和 B 分别是类名称，他们都继承了 object 类（这个是基类，Python 中所有的 class 都是从 object 继承而来的，它也可以不写）。和函式一样，class 也可以定义文档字符串，同样是通过__doc__访问。和大多数编程语言不同的是，Python 中构造函式叫做__init__，第一个参数传递的就是将要初始化的实例对象本身，类似许多语言中的 this。同样，Python 也有类似的析构函式叫做__del__。

构造函式中的 self.a = a 便给 self 这个对象增加了一个名为 a 的属性，其绑定的对象就是传入的参数 a 所绑定的对象。对象的__dict__属性可以查看对象的所有属性。在任何时候都可以给对象增加属性，只要给不存在的属性绑定对象即可，Python 会自动创建不存在的属性。

类也可以有自己的属性。

```
In [4]: class A(object):
   ...:     """This is Class A"""
   ...:     aa = 'sth'
   ...:     def __init__(self, a):
   ...:         self.a = a
   ...:

In [5]: A.aa
Out[5]: 'sth'
```

同样也可以通过类名称、类属性名的形式访问类属性。

类中的方法和普通的函式定义是类似的。

```
In [7]: class A(object):
   ...:     """This is Class A"""
   ...:     aa = 'sth'
   ...:     def __init__(self, a):
   ...:         self.a = a
   ...:     def do_sth(self):
   ...:         pass
```

类继承

Python 中允许单继承和有限的多继承。

```
In [13]: class C(A, B):
   ....:   def __init__(self, a, b, c):
   ....:       A.__init__(self,a)
   ....:       B.__init__(self,b)
   ....:       self.c = c
   ....:

In [18]: objc = C(1, 2, 3)

In [19]: objc.__dict__
Out[19]: {'a': 1, 'b': 2, 'c': 3}
```

如果所需要的一个属性在类 C 中没有找到，则从左到右依次按 A，B 的顺序在基类中寻找。通过`__dict__`可以看到对象的所有属性。

小结

类的定义和使用在面向对象编程中是十分重要的内容，应该熟练掌握。有关类和面向对象的进一步资料如下所示。

- 类与实例：*http://wiki.woodpecker.org.cn/moin/ObpLovelyPython/LpyQLearn-5-object*
 精巧地址：*http://bit.ly/36elwi*
- 对象和面向对象：
 http://www.woodpecker.org.cn/diveintopython/object_oriented_framework/index.html#fileinfo.divein
 精巧地址：*http://bit.ly/1WOEJk*
- A First Look at Classes：*http://www.python.org/doc/2.6/tutorial/classes.html*
 精巧地址：*http://bit.ly/3VhOJ3*

PCS112　异常

概述

使用 try...except 可以捕捉各种异常类型。Python 中有很多内置的异常类型，具体可以参见下面的资料。

- Python 异常类型：*http://www.python.org/doc/current/lib/module-exceptions.html*
 精巧地址：*http://bit.ly/rI7AI*

应用

捕捉异常

首先举个小例子。

```
try:
    code_block()
    #...
except SomeException, e:
    do_some_thing_with_exception(e)
except (Exception1, Exception2), e:
    do_some_thing_with_exception(e)
except:
```

```
 9    do_some_thing_with_other_exceptions()
10 else:
11    do_some_thing_when_success()
12 finally:
13    do_some_thing()
```

try...except 可以带一个 else 子句，该子句只能出现在所有 except 子句之后。当 try 语句没有抛出异常时，需要执行一些代码，可以使用这个子句。无论 try 子句中有没有发生异常，finally 子句都一定会被执行。如果发生异常，在 finally 子句执行完后它会被重新抛出。try 子句经由 break 或 return 退出也一样会执行 finally 子句。

抛出异常

使用 raise 来抛出异常。

```
1 try:
2    raise NameError, 'HiThere'
3 except NameError, a:
4    print 'An exception flew by!'
5    print type(a)
```

raise 函式的第一个参数是异常名，第二个是这个异常的实例，它存储在 instance.args 的参数中。和 except NameError, a: 中的第二个参数意思差不多。

自定义异常

异常类中可以定义任何其他类中可以定义的东西，但是通常为了保持简单，只在其中加入几个属性信息，以供异常处理句柄提取。

```
1  class MyError(Exception):
2    def __init__(self, value):
3       self.value = value
4    def __str__(self):
5      return repr(self.value)
6 try:
7    raise MyError(2*2)
8 except MyError, e:
9    print 'My exception occurred, value:', e.value
```

如果一个新创建的模块中须抛出几种不同的错误时，一个通常的做法是为该模块定义一

个异常基类，然后针对不同的错误类型派生出对应的异常子类。

小结

本文介绍了 Python 中如何捕捉异常、抛出异常和如何自定义异常来方便编程. 在下面的参考链接中可以找到更多关于异常的知识。

- 异常处理：*http://wiki.woodpecker.org.cn/moin/ObpLovelyPython/LpyQLearn-3-process#id2*
 精巧地址：*http://bit.ly/1o9j0x*
- try ... except ... finally / raise：
 http://wiki.woodpecker.org.cn/moin/PyAbsolutelyZipManual#head-263e21b75a15eebdaa6574be53cfbddebb7a1231
 精巧地址：*http://bit.ly/1BfHoi*
- Python 学习笔记：*http://www.blogjava.net/JAVA-HE/archive/2008/04/10/191867.html*
 精巧地址：*http://bit.ly/JCokE*

PCS113　交互参数

概述

Python 中的 sys 模块提供一些系统变量和函式。其中的 `argv` 代表调用 Python 脚本时的命令行参数列表。argv[0]是脚本的名称（由于操作系统的不同而显示绝对路径名或者相对路径名），其后为各命令行的选项或参数。

```
In [1]: import sys

In [2]: len(sys.argv)
Out[2]: 1

In [3]: print sys.argv
['/usr/bin/ipython']
```

应用

Python 中提供的很多模块都可以解析系统参数，比如，cmd、getopt 和 optparse.

- Cmd：*http://docs.python.org/lib/module-cmd.html*
 精巧地址：*http://bit.ly/4y8FC9*
- getopt：*http://docs.python.org/lib/module-getopt.html*
 精巧地址：*http://bit.ly/4kIlCa*
- optparse：*http://docs.python.org/lib/module-optparse.html*

精巧地址：*http://bit.ly/1XZtW8*

下面举个 optparse 的简单例子：

```
 1 from optparse import OptionParser
 2 import sys
 3
 4 #...
 5 def execute_from_command_line(argv=None):
 6    if argv is None:
 7       argv = sys.argv
 8    # Parse the command-line arguments.
 9    parser = OptionParser(usage="""my_html_to_txt.py [options] xxx""")
10    parser.add_option('-s', '--search', help='Search keywords frequence',
       dest='keyword')
11    parser.add_option('-c', '--create', help='Create a txt file from many
       html', action='store_true', dest='create')
12    parser.add_option('-f', '--frequence', help='Get Words Frequence',
       action='store_true', dest='frequence')
13    parser.add_option('-d', '--delete', help='delete Http',
      action='store_true', dest='delete')
14    #...
15
16    options, args = parser.parse_args(argv[1:])
17    if options.keyword:
18       print options.keyword
19    if options.create:
20       print 'create sth'
21    #...
22
23 if __name__ == '__main__':
24    execute_from_command_line(sys.argv)
```

首先声明一个 OptionParser 对象，然后通过 add_option 增加选项，根据这些选项做出不同的动作，完成不同的功能。最终效果如下：

```
~$ python pcs-113-1.py -s key -c
key
create sth
```

PCS114　FP 初体验

概述

Python 不是一门纯粹的函式编程语言，它也加入了一些可以有效提高编程效率的函式编程特性。

函式编程，英文写作 Functional Programming，原意并非以函式来组织程序结构，直白地说，FP 是指将程序的行为抽象出来，作为基本的元素，以此组织程序。它来自于与图灵机同时提出的 Lambda 计算模型。相对于具有高度可操作性（现代我们所见的计算机基本都是图灵机）的图灵机模型，Lambda 模型在数学上更为优美，因此，产生了 FP 语言这样的技术流派，通过严格定义程序的行为来使程序可控性更好，更为优美。相对来说，FP 语言也须要较深入地学习。

应用

在函式编程中，最著名的特色就是高序（High Order）。简单地说，就是定制一个算法，按规则来指定容器中的每一个元素。最常用的 High Order 为：

- 映射，也就是将算法施于每个元素，将返回值合并为一个新的容器。
- 过滤，将算法施于每个元素，将返回值为真的元素合并为一个新的容器。
- 合并，将算法（可能携带一个初值）依次施于每个元素，将返回值作为下一步计算的参数之一，与下一个元素再计算，直至最终获得一个总的结果。

Python 通过 map、filter、reduce 三个内置函式来实现这三个 High Order 功能。

使用 map

函式 map 至少需要两个参数，第一个是一个函式，第二个是传入函式的参数。例如：

```
1
2 def foo(x):
3    return x*x
4
5 print map(foo, range(10))
```

以上代码得到 10 以内的自然数平方表。

```
~$ python pcs-114-1.py
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

map 允许接收三个或三个以上的参数，从第二个开始，每个参数都接收一个线性容器（或迭代对象），将每个元素提取出来作为第一个参数（函式）的参数列表。如果各个容器的长度不一样，短缺的部分用 None 补齐。例如可以这样使用：

```
1
2 def foo(x, y):
3    return x**y
4
5 print map(foo, range(10), range(10))
```

```
~$ python pcs-114-2.py
[1, 1, 4, 27, 256, 3125, 46656, 823543, 16777216, 387420489]
```

这个计算结果会很大，所以最好不要用太大的数来测试，不过得益于 Python 的长整数，仍然可以得到计算结果。如果直接这样用：map(foo, range(10), range(20))，会收到异常，因为定义的这个 foo 不接受 None。

如果 map 的第一个参数为 None，那么返回原来的序列，如果传入了多个序列，会将其中每个位置的参数打包成 tuple。如：

```
1
2 print map(None, range(10), range(10))
```

```
~$ python pcs-114-3.py
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8),
(9, 9)]
```

使用 filter

英文单词 filter 的字面意义有过滤的意思，实际也如此。例如可以用下面的方法得到 100 以内的偶数列：

```
1
2 def foo(x):
3    return x%2==0
4
5 print filter(foo, range(100))
```

```
~$ python pcs-114-4.py
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38,
40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76,
78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98]
```

如果 filter 的第一个参数是 None，则返回序列中所有的真值。例如 True、非空序列，非零数值等。

使用 reduce

filter 可以方便地实现有条件的选择，但是如果想要用筛法计算素数列，用 filter 就不够高效了。计算 N 是否为素数最有效的方法应该只计算已得到的素数列中，小于 N 的平方根的那部分。而 filter 并不保存前一步的计算状态（从 FP 理论上讲，每个 filter 运算之间是无关的，它们可以并行执行）。需要将上一步计算的结果作为当前计算的一部分时，Python 的内置函式 reduce 就派上用场了。

```
 1
 2 def foo(perms, x):
 3    i = 0
 4    while perms[i]**2 <= x:
 5       if x%perms[i] == 0:
 6          return perms
 7       else:
 8          i += 1
 9    else:
10       perms.append(x)
11    return perms
12
13 print reduce(foo, range(5, 100, 2), [2, 3])
```

```
~$ python pcs-114-5.py
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97]
```

因为 1、2、3、5 这几个数已经确定为素数，并且大于 2 的偶数也显然不是素数，所以这里对传统筛选法做了很普通的优化。可以看到，通过 reduce，方便地实现了计算结果的重用，节省了大量的 CPU 周期。在实际应用中，reduce 可以用来实现统计计算或时序依赖的遍历行为。

Lambda 表达式

FP 的理论基础，称之为 Lambda 模型。在很多 FP 语言中，Lambda 这个伟大的名字就代表了函式。而 Python 中，Lambda 是一个关键字，是一个简单的匿名函式定义方式。它允许将一个表达式定义为一个可调用的对象，可以用它赋值或传递给其他函式。

前面讨论了一个通过 map 生成乘方表的例子，实际上那个例子中的 foo 函式只有一行，完全可以用 Lambda 代替它：

```
print map(lambda x:x**2, range(10))
```

```
~$ python pcs-114-6.py
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Python 中的 Lambda 不像在真正的 FP 语言中那么显要，它只是一个语法糖，用来生成一个匿名的函式对象。Lambda 只接受一个单行表达式，该表达式的结果就是返回值。Python 的设计者出于可读性和语法的一致性考虑，没有支持多行匿名函式的定义，但是 Python 可以嵌套定义函式。

推导式

这项技术在 Python 中的正式命名是“List Comprehensions”，在 Python Tutorial 的简体中文版中译作列表推导式。其基本形式很简单，例如，前面生成平方表的例子还可以进一步简化为：

```
1
2 print [x**2 for x in range(10)]
```

```
~$ python pcs-114-7.py
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

列表推导式分为三部分，最左边是生成每个元素的表达式，然后是 for 迭代过程，最右边可以设定一个 if 判断作为过滤条件。例如通过下面的方式查找 1000 以内，所有左右对称的数：

```
 1
 2 def isSymmetry(it):
 3     if it < 10:
 4         return True
 5     s = str(it)
 6     mpoint = len(s)/2 + 1
 7     for idx in range(1, mpoint):
 8         if s[idx-1] != s[-idx]:
 9             return False
10     else:
11         return True
12
13 print [x for x in range(1000) if isSymmetry(x)]
```

```
~$ python pcs-114-8.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 22, 33, 44, 55, 66, 77, 88, 99, 101, 111,
121, 131, 141, 151, 161, 171, 181, 191, 202, 212, 222, 232, 242, 252, 262,
272, 282, 292, 303, 313, 323, 333, 343, 353, 363, 373, 383, 393, 404, 414,
424, 434, 444, 454, 464, 474, 484, 494, 505, 515, 525, 535, 545, 555, 565,
575, 585, 595, 606, 616, 626, 636, 646, 656, 666, 676, 686, 696, 707, 717,
727, 737, 747, 757, 767, 777, 787, 797, 808, 818, 828, 838, 848, 858, 868,
878, 888, 898, 909, 919, 929, 939, 949, 959, 969, 979, 989, 999]
```

列表推导式甚至还可以一次性执行多个 for 迭代，这会生成它们的全排列，例如可以通过以下方法生成一个九九乘法表：

```
1
2 for s in ["%s * %s = %s"%(x, y, x*y) for x in range(1, 10) for y in
  range(1, 10)]:
3     print s
```

```
~$ python pcs-114-9.py
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
...
8 * 8 = 64
8 * 9 = 72
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
```

如果只希望输出不重复的那些，不妨尝试以下代码：

```
1
2 for s in ["%s * %s = %s"%(x, y, x*y) for x in range(1, 10) for y in
  range(1, 10) if x<=y]:
3    print s
```

```
~$ python pcs-114-10.py
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
2 * 2 = 4
```

```
2 * 3 = 6
2 * 4 = 8
...
8 * 8 = 64
8 * 9 = 72
9 * 9 = 81
```

因为列表推导式可以统一实现 map 和 filter，而 reduce 可以转成等价的 for，Python 的发明人 Gudio 曾经想取消 map、filter 和 reduce 这三个函式。虽然在广大使用者的呼吁下这三个函式还是保留了下来，但是大家也一致认可，列表推导通常可以用更简洁的表达从线性容器中有选择地获取并计算元素，所以应该优先使用列表推导式。

因为列表推导式非常好用，所以在 Python 2.5 之后，对它做了进一步的扩展，如果一个函式接受一个可迭代对象作为参数，那么可以给它传递一个不带中括号的推导式，在 Python Tuorial 中称其为“迭代推导式”，因为它不需要一次生成整个列表，只要将可迭代对象传递给函式，如果推导式返回大量元素，这样做显然可以节省内存，提高速度。例如字典的构造函式可以接受一个可迭代对象，从中得到（key, value）元组序列来生成字典，在使用 Python 2.5 时，经常可以见到形如以下的代码：

```
1
2 d = dict(("%s * %s"%(x, y), x * y) for x in range(1, 10) for y in
  range(1, 10))
3 for k, v in d.iteritems():
4    print "%s = %s"%(k, v)
```

```
~$ python pcs-114-11.py
8 * 4 = 32
7 * 4 = 28
5 * 2 = 10
6 * 4 = 24
4 * 2 = 8
4 * 6 = 24
8 * 7 = 56
1 * 6 = 6
5 * 6 = 30
5 * 9 = 45
```

```
...
4 * 3 = 12
8 * 3 = 24
9 * 9 = 81
8 * 8 = 64
8 * 2 = 16
```

模块篇

PCS200	os(.stat;.path)	258
PCS201	cmd	264
PCS202	chardet	267
PCS203	epydoc	270
PCS204	ConfigParser	274
PCS205	内建函式(enumerate)	277
PCS206	thread	280
PCS207	threading	282
PCS208	dict4ini	285
PCS209	fnmatch	288
PCS210	pickle	290
PCS211	base64	294
PCS212	shutil	298
PCS213	time	304
PCS214	ElementTree	309
PCS215	random	312
PCS216	socket	315
PCS217	Tkinter	319

PCS200　os(.stat;.path)

概述

os.path 是一个与平台无关的文件路径处理模块。它可以帮助我们解决在程序处理中碰到的一些复杂的路径处理问题，如“//”、“\”、“\\”、“.” 等路径分隔符处理不全面的问题，跨平台问题和组合一个可用的跨平台的路径地址问题。我们用简单的字符串拆分这些路径处理问题是很困难或者无法完成的，但是 os.path 都替你做好了解决方案，你只须使用它里面的函式就可以了。

应用

os.path.split

os.path.split 的函式功能：os 拆分路径，返回一个 tuple，第一个元素是文件所在路径，第二个元素是对应文件名。

如下面的一个小测试：

```
1 import os.path
2
3 for path in [ '/one/two/three',
4             '/one/two/three/',
5             '/',
```

```
6             '.',
7             '']:
8    print '"%s" : "%s"' % (path, os.path.split(path))
```

对于一个文件（具有绝对路径或相对路径），使用 os.path.split()可以将其拆成对应路径名（不以路径分割符（如'/'结尾））和文件名；对于一个路径，则返回该路径（同样，不以路径分割符，（如'/'结尾）），而对应的文件名为空。这里比较奇特的是对于当前文件夹('.')，os.path.split()把它看作是文件名。

```
~$ python pcs-200-1.py
"/one/two/three" : "('/one/two', 'three')"
"/one/two/three/" : "('/one/two/three', '')"
"/" : "('/', '')"
"." : "('', '.')"
"" : "('', '')"
```

os.path.basename

os.path.basename 的函式功能：只获取某路径对应的文件名。

修改上面的例子：

```
1 import os.path
2
3 for path in [ '/one/two/three',
4              '/one/two/three/',
5              '/',
6              '.',
7              '']:
8    print '"%s" : "%s"' % (path, os.path.basename(path))
```

如果这里的函式参数只是一个路径，不是文件名，则返回对应的文件名为空。同样对于'.'也比较奇怪。

```
~$ python pcs-200-2.py
"/one/two/three" : "three"
"/one/two/three/" : ""
"/" : ""
"." : "."
"" : ""
```

os.path.dirname

os.path.dirname 的函式功能：只获取某路径对应的路径，不含文件名。

再修改之前的例子：

```
1 import os.path
2
3 for path in [ '/one/two/three',
4              '/one/two/three/',
5              '/',
6              '.',
7              '']:
8     print '"%s" : "%s"' % (path, os.path.dirname(path))
```

同样，返回的路径名是不以路径分割符（如'/'）结尾的。

```
~$ python pcs-200-3.py
"/one/two/three" : "/one/two"
"/one/two/three/" : "/one/two/three"
"/" : "/"
"." : ""
"" : ""
```

os.path.splitext

os.path.splitext 的函式功能：将路径、文件名、扩展名分开，并以一个 tuple 的形式返回。

```
1 import os.path
2
3 for path in [ 'filename.txt', 'filename', '/path/to/filename.txt', '/', '' ]:
4     print '"%s" :' % path, os.path.splitext(path)
```

可以看到 os.path.splittext 只是很单纯地将文件名和扩展名分开了，如下所示。

```
~$ python pcs-200-4.py
"filename.txt" : ('filename', '.txt')
"filename" : ('filename', '')
"/path/to/filename.txt" : ('/path/to/filename', '.txt')
"/" : ('/', '')
"" : ('', '')
```

os.path.commonprefix

os.path.commonprefix 的函式功能：看一个比较 cool 的功能就是要在一组路径中，找到一个共同的前缀，比如：

```
1 import os.path
2
3 paths = ['/one/two/three/four',
4         '/one/two/threefold',
5         '/one/two/three/',
6         ]
7 print paths
8 print os.path.commonprefix(paths)
```

```
~$ python pcs-200-5.py
['/one/two/three/four', '/one/two/threefold', '/one/two/three/']
/one/two/three
```

os.path.join

os.path.join 的函式功能：使用 os.path.join 组合一些零散的字符串，生成一个安全的路径表示，如下所示。

```
1 import os.path
2
3 for parts in [ ('one', 'two', 'three'),
4              ('/', 'one', 'two', 'three'),
5              ('/one', '/two', '/three'),
6              ]:
7    print parts, ':', os.path.join(*parts)
```

也就是将一系列短路径拼合成有效的长路径。

```
~$ python pcs-200-6.py
('one', 'two', 'three') : one/two/three
('/', 'one', 'two', 'three') : /one/two/three
('/one', '/two', '/three') : /three
```

os.path.expanduser

os.path.expanduser 的函式功能：可以利用 os.path.expanduser 寻找用户的 home 目录，如下

所示。

```
1 import os.path
2
3 for user in [ '', 'root', 'mysql' ]:
4    lookup = '~' + user
5    print lookup, ':', os.path.expanduser(lookup)
```

'~user'是表示某个用户的 home 目录，'~'表示当前用户的 home 目录。

```
~$ python pcs-200-7.py
~ : /home/shengyan
~root : /root
~mysql : /var/lib/mysql
```

os.path.expandvars

os.path.expandvars 的函式功能：使用 os.path.expandvars 来读取路径中系统环境变量的值。在下面的示例中，先使用 os.environ 增加定义了一个环境变量'MYVAR'，并为其赋值'VALUE'，然后使用 os.path.expandvars 将出现在路径中的环境变量扩展为对应值。

```
1 import os.path
2 import os
3
4 os.environ['MYVAR'] = 'VALUE'
5
6 print os.path.expandvars('/path/to/$MYVAR')
```

```
~$ python pcs-200-8.py
/path/to/VALUE
```

os.path.normpath

os.path.normpath 的函式功能：处理不规则路径字符串，将其转化为正常的路径。比如：

```
1 import os.path
2
3 for path in [ 'one//two//three',
4             'one/./two/./three',
5             'one/../one/two/three',
6             ]:
7    print path, ':', os.path.normpath(path)
```

```
~$ python pcs-200-9.py
one//two//three : one/two/three
one/./two/./three : one/two/three
one/../one/two/three : one/two/three
```

os.path.abspath

os.path.abspath 的函式功能：将相对路径转换为绝对路径，如下所示。

```
1 import os.path
2
3 for path in [ '.', '..', './one/two/three', '../one/two/three']:
4    print '"%s" : "%s"' % (path, os.path.abspath(path))
```

```
~$ python pcs-200-10.py
"." : "/home/shengyan/LovelyPython/PCS/pcs-200"
".." : "/home/shengyan/LovelyPython/PCS"
"./one/two/three" :
"/home/shengyan/LovelyPython/PCS/pcs-200/one/two/three"
"../one/two/three" : "/home/shengyan/LovelyPython/PCS/one/two/three"
```

问题

要注意在两个根目录做 join 操作时经常出现的问题，如下所示：

```
1 import os
2 print os.path.join('/tmp','/var')
```

执行结果为：

```
'/var'
```

从结果中可以看到，并不是将两个路径做了简单连接，通过看 help("os.path.join")可以看到，路径的拼接是从/开始的，所以会以“后一个开始/”为初始路径，要特别注意。

探讨

如果不使用 os.path 这个模块，而是自己手工处理以上的例子，可以试试看需要如何处理，会遇到哪些困难。

PCS201　cmd

概述

cmd 模块为命令行接口（command-line interfaces ， CLI）提供了一个简单的框架。 它经常被用在 pdb（Python 调试模块）模块中，当然也可以在自己的程序中使用它来创建命令行程序。

应用

下面先举个 cmd 模块使用的小例子：

```
 1 # -*- coding: utf-8 -*-
 2 import cmd
 3 import string, sys
 4
 5 class CLI(cmd.Cmd):
 6
 7    def __init__(self):
 8       cmd.Cmd.__init__(self)
 9       self.prompt = '> '    # 定义命令行提示符
10
11    def do_hello(self, arg):   # 定义 hello 命令所执行的操作
12       print "hello again", arg, "!"
13
14    def help_hello(self):       # 定义 hello 命令的帮助输出
```

```
15        print "syntax: hello [message]",
16        print "-- prints a hello message"
17
18    def do_quit(self, arg):     # 定义 quit 命令所执行的操作
19        sys.exit(1)
20
21    def help_quit(self):        # 定义 quit 命令的帮助输出
22        print "syntax: quit",
23        print "-- terminates the application"
24
25    # 定义 quit 的快捷方式
26    do_q = do_quit
27
28 # 创建 CLI 实例并运行
29 cli = CLI()
30 cli.cmdloop()
```

从这个例子可以看出，首先 CLI 类继承了 cmd.Cmd 类，然后在类中定义了两条命令 hello 和 quit，而命令 q 被作为 quit 的短命令形式。也就是说，若须另外定义一条命令，如 command，只要在 CLI 类中增加一个 do_command 函式，而该命令对应的帮助信息由 help_command 函式给出。

使用 `cmd.Cmd` 类编写命令行处理程序是非常容易的。运行上述例子，可以进入如下的命令行：

```
~$ python pcs-201.py
> ?

Documented commands (type help <topic>):
========================================
hello  quit

Undocumented commands:
======================
help  q

> help hello
syntax: hello [message] -- prints a hello message
> hello LovelyPython
hello again LovelyPython !
> find
*** Unknown syntax: find
> q
```

就像示例中所写的那样，自定义的 CLI 类提供了 `hello` 和 `quit` 命令，可以正常使用它们，而 `find` 命令是没有定义的，所以命令行提示为未知语法。最后的 `q` 命令和 quit 是一样的功能，即退出程序。

小结

使用 cmd 模块可以方便编写命令行程序，同时使用 getopt 和 optparse 这两个模块可以很方便地解析命令行参数。这里向读者推荐一个比 cmd 更好的模块，是由 Doug Hellmann 编写的命令行处理类 `CommandLineApp`，具体可以访问以下网站：

- CommandLineApp 描述：

 http://www.doughellmann.com/articles/CommandLineApp/index.html

 精巧地址： *http://bit.ly/1b2tsB*
- CommandLineApp 相关代码：*http://code.pythonmagazine.com/2/1*

 精巧地址：*http://bit.ly/1aUqx4*

PCS202　chardet

概述

chardet 是一个开源的字符编码自动检测模块，是基于 Python 实现的。读者可以在 chardet 的网站上下载这个模块并根据源码包中的相关说明进行安装。

- chardet 模块下载：*http://chardet.feedparser.org/download/chardet-1.0.1.tgz*
- 精巧地址：*http://bit.ly/8EXnf*

应用

检测字符编码

利用 chardet 来自动检测字符编码的最简单方式是使用 dectect()，它以一个非 unicode 字符串作为参数，返回一个由被检测字符串的编码方式及其可信度数值（介于 0~1 之间）组成的字典数据结构。下面的例子中，首先使用 urllib.urlopen 方法打开百度首页，并读取其内容作为被检测数据，然后使用 chardet.detect 检测并返回结果：

```
>>> import urllib
>>> rawdata = urllib.urlopen('http://www.baidu.com').read()
>>> import chardet
>>> chardet.detect(rawdata)
{'confidence': 0.98999999999999999, 'encoding': 'GB2312'}
```

可以看到，检测出该网页编码方式 GB2312 的可信度是 0.98999999999999999，一个非常高的值。

渐进检测字符编码

对于大量的文本，chardet 提供了可以渐进检测字符编码的相关方法，并且在满足可信度后会自动停止检测。具体是通过 UniversalDetector 类的 feed 方法不断检测每个文本块，当达到最小可信度阈值时，就标记 UniversalDetector 类的 done 值为 True，表示完成检测。已经读取完待检测文本之后若调用 UniversalDetector 类的 close 方法，它会在没有达到最小可信度的情况下再做一些计算，以便返回最大程度上准确的值，该值和 chardet.detect 函式是一样的字典结构。

下面是一个例子：

```
 1 import urllib
 2 from chardet.universaldetector import UniversalDetector
 3
 4 usock = urllib.urlopen('http://www.baidu.com')
 5 detector = UniversalDetector()
 6 for line in usock.readlines():
 7    detector.feed(line)
 8    if detector.done: break
 9 detector.close()
10 usock.close()
11 print detector.result
```

这个例子同样先打开百度首页，接着创建了一个 UniversalDetector 对象 detector，在接下来的 for 循环中，读取数据的同时检测该数据，当 done 值为 True 时表示检测完成退出 for 循环，最后关闭数据流并打印结果：

```
~$ python pcs-202-1.py
{'confidence': 0.98999999999999999, 'encoding': 'GB2312'}
```

如果想对多个独立文本进行编码检测，可以重复使用同一个 UniversalDetector 对象，只要在每个文本开始检测前调用 reset()，以表明接下来读取的是与之前文本相独立的字符串，然后就可以在类似上述例子中调用 feed()，最后调用 close()结束，最终的检测结果放在 result 中，如下所示。

```
 1 import glob
 2 from chardet.universaldetector import UniversalDetector
 3
```

```
 4 detector = UniversalDetector()
 5 for filename in glob.glob('*.py'):
 6    print filename.ljust(60),
 7    detector.reset()
 8    for line in file(filename, 'rb'):
 9       detector.feed(line)
10       if detector.done: break
11    detector.close()
12    print detector.result
```

其运行结果如下：

```
~$ python pcs-202-2.py
pcs-202-1.py              {'confidence': 1.0, 'encoding': 'ascii'}
pcs-202-2.py              {'confidence': 1.0, 'encoding': 'ascii'}
```

PCS203　epydoc

概述

epydoc 是专门用于从源码注释中生成各种格式（如 Html、plaintext、LaTeX 和 PDF）文档的工具。它支持多种文档化标签语法：Epytext（轻量级的标记语言，用于标记各种文档字符串，主要是在一些特定的标签上添加相关的信息，例如参数类型等）、ReStructuredText（一种易用的，所见即所得的文本标签语法）、Javadoc（用于 Java 源代码的文档化标记语言）以及普通文本。

应用

安装

这里先介绍源码安装的步骤：

```
~$ wget http://prdownloads.sourceforge.net/epydoc/epydoc-3.0.1.tar.gz
#下载
~$ tar zxvf epydoc-3.0.1.tar.gz #解压文件
~$ cd epydoc-3.0.1/ #进入目标目录
~/epydoc-3.0.1$ sudo python setup.py install #安装，需要 root 权限
[sudo] password for shengyan:
running install
running build
running build_py
```

```
...
#完毕
```

其他平台下的更多安装方法可参见安装手册：

http://epydoc.sourceforge.net/manual-install.html

精巧地址：*http://bit.ly/2lsakW*

书写格式

使用 epydoc 生成文档，得先在源代码中插入相关的各种格式的 Docstring，它支持多种标签。Epydoc 支持标签的详细介绍请看：*http://wiki.woodpecker.org.cn/moin/EpydocApiTag*，精巧地址：*http://bit.ly/2JBwrl*

以下列出基本的标签格式。

1. py 文献信息

```
'''@author:''' ... 作者
'''@license:''' ... 版权
'''@contact:''' ... 联系
```

2. py 状态信息

```
'''@version:''' ... 版本推荐使用$Id$
'''@todo '''[ver]: ... 改进，可以指定针对的版本
```

3. py 模块信息

```
'''@var '''v: ... 模块变量 v 说明
'''@type '''v: ... 模块变量类型 v 说明
```

4. py 函式信息

```
'''@param '''p: ... 参数 p 说明
'''@type '''v: ... 参数 p 类型说明
'''@return: ''' ... 返回值说明
'''@rtype '''v: ... 返回值类型说明
```

5. py 提醒信息

```
'''@note:''' ... 注解
'''@attention:''' ... 注意
'''@bug:''' ... 问题
'''@warning:''' ... 警告
```

6. py 关联信息

```
'''@see:''' ... 参考资料
```

epydoc 支持三种标签的语法。

- Epytext

```
@tag: 内容...
```

- ReStructuredText

```
:tag: 内容...
```

- Javadoc

```
@tag 内容...
```

更多的 Tag 语法及注释规范可参见文档化开发注释规范：

http://wiki.woodpecker.org.cn/moin/CodeCommentingRule

精巧地址：*http://bit.ly/46BkrB*

文档生成

功能描述：将 docstring 生成 html 文档

```
1 # coding: utf-8
2 def MyFunc():
3     '''
4     docstring 文档
5     用 epydoc 来生成我
6     '''
7     pass
```

运行 epydoc 来生成文档：

```
~$ epydoc pcs-203-1.py
```

在当前目录下会生成一个 html 文件夹，点击 index.html，即可看到生成的文档结果（如图 PCS203-1 所示）：

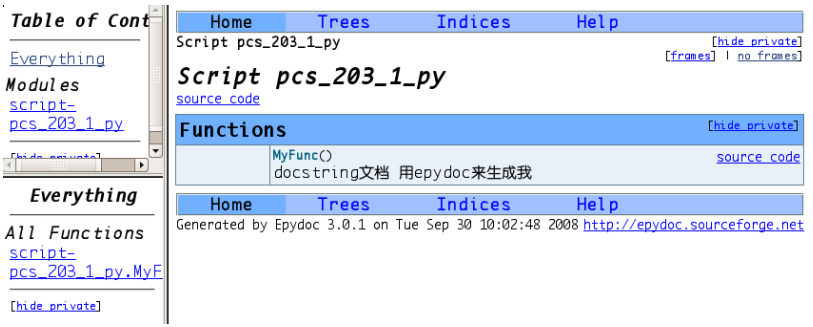


图 PCS203-1

问题

还可以通过书写一个简单的配置文件，生成更丰富的文档。如：

```
##可以命名为 epydoc.cfg
[epydoc]
# 项目信息，名称及 URL
name: Backup
url: http://wiki.woodpecker.org.cn/moin/ObpLovelyPython
#指定想要处理的模块
modules: main.py, inotify.py
#指定输出形式及位置
output: html
target: apidocs/
#指定图形生成工具，这样就可以产生各种 UML 图
graph: all
dotpath: /usr/bin/dot
```

然后使用如下命令即可生成所需文档：

```
~$ epydoc --config epydoc.cfg  #获得最新的 API 文档
```

PCS204　ConfigParser

概述

ConfigParser 是用来处理 ini 格式的配置文件的 Python 标准库。

为自己的程序建立一个可配置的文件是一个良好的习惯，可以不修改源代码，就能动态地改变程序的运行结果，方便维护。

应用

read(self, filenames)

read(self, filenames)的函式功能：直接读取 ini 文件内容，filenames 可以是一文件名或是文件名列表。

示例如下：

```
>>> from ConfigParser import ConfigParser
>>> config = ConfigParser()
>>> config.read('approachrc')
['approachrc']
```

sections(self)

sections(self)的函式功能：得到所有的 section，并以列表的形式返回。

示例如下：

```
>>> config.sections()
['portal']
```

options(self, section)

options(self, section)的函式功能：得到指定 section 的所有 option。

示例如下：

```
>>> config.options('portal')
['username', 'host', 'url', 'password', 'port']
```

get(self, section, option, raw=False, vars=None)

get(self, section, option, raw=False, vars=None)的函式功能：得到 section 中 option 的值，返回为 string 类型。

示例如下：

```
>>> config.get('portal', 'username')
'dhellmann'
```

set(self, section, option, value)

set(self, section, option, value)的函式功能：对 section 中的 option 进行设置，其值为 value。

示例如下：

```
>>> config.set('portal', 'username', 'lovelypython')
>>> config.get('portal', 'username')
'lovelypython'
```

实例

假设有如下的配置文件示例：

```
[portal]
url = http://%(host)s:%(port)s/Portal
```

```
username = dhellmann
host = localhost
password = SECRET
port = 8080
```

使用 ConfigParse 进行解析：

```
 1 from ConfigParser import ConfigParser
 2 import os
 3
 4 filename = os.path.join('.', 'approachrc')
 5 print filename
 6
 7 config = ConfigParser()
 8 config.read(filename)
 9
10 url = config.get('portal', 'url')
11 print url
```

这段代码中，config 读取 approachrc 内容，获得字段 portal 中选项 url 的值，可以看到 approachrc 中的 url 为 http://%(host)s:%(port)s/Portal，这里的%(host)s 和%(port)s 又分别读取了 host 和 port 的值，最终得到了如下的结果：

```
~$ python pcs-204-1.py
./approachrc
http://localhost:8080/Portal
```

PCS205　内建函式(enumerate)

概述

Python 除了有语言简洁，容易上手等优点，还有一个重要的优点，就是存在大量的内置函式，方便编程。本将介绍这些常用函式，让我们更好地了解 Python 的诱人之处。

应用

enumerate

enumerate 是 Python 2.3 中新增的内置函式，它的英文说明为:

```
enumerate(iterable)
   Return an enumerate object. iterable must be a sequence, an iterator,
or some other object which supports iteration. The next() method of the
iterator returned by enumerate() returns a tuple containing a count (from
zero) and the corresponding value obtained from iterating over iterable.
enumerate() is useful for obtaining an indexed series: (0, seq[0]), (1,
seq[1]), (2, seq[2]), .... New in version 2.3.
```

它特别适合用于 for 循环，当我们同时需要序号和元素时可以使用这个函式。

比如，有一个字符串数组，需要一行一行打印出来，同时每行前面加上序号，从 1 开始，如下所示：

```
1 mylist=["a","b","c"]
2 for index,obj in enumerate(mylist):
3    print index,obj
```

输出结果为：

```
0 a
1 b
2 c
```

map

函式说明：map(function, sequence[, sequence, ...]) -> list，这两个参数中一个是函式名，另一个是列表或元组，它会返回一个列表。比如，将数组中每一个数乘以 2：

```
1 print "map(lambda x:x*2,[1,2,3,4,5]) -> ",map(lambda x:x*2,[1,2,3,4,5])
```

输出结果为：

```
map(lambda x:x*2,[1,2,3,4,5]) ->  [2, 4, 6, 8, 10]
```

zip

函式说明：zip(seq1 [, seq2 [...]]) -> [(seq1[0], seq2[0] ...), (...)]，这个函式可以同时循环两个一样长的数组，返回一个包含每个参数元组对应元素的元组。各参数元组长度最好一致，若不一致，采取截断方式，使得返回的结果元组的长度为各参数元组长度最小值。比如：

```
1 print "zip([1,2,3],[4,5,6]):"
2 for x,y in zip([1,2,3],[4,5,6]):
3    print "x,y",x,y
```

输出结果为：

```
zip([1,2,3],[4,5,6]):
x,y 1 4
x,y 2 5
x,y 3 6
```

filter

函式说明：filter(function or None, sequence) -> list, tuple, or string，它包括两个参数，分别是 function 和 list。该函式根据 function 参数返回的结果是否为真来过滤 list 参数中的项，

最后返回一个新列表。比如，过滤掉数组中小于 3 的数：

```
1 print "filter(lambda x:x>3,[1,2,3,4,5]) -> ",filter(lambda
  x:x>3,[1,2,3,4,5])
```

输出结果为：

```
filter(lambda x:x>3,[1,2,3,4,5]) ->  [4, 5]
```

dir

函式说明：它用于列出一个变量的所有方法和属性。

动态语言经常遇到的问题就是，当前的变量究竟有哪些可用的方法和属性？通过 dir()函式就可以很容易地解决了，如下所示：

```
1 s="Hello Python"
2 print dir(s)
```

运行结果：

```
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
'__eq__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__',
'__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__',
'__str__', 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith',
'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',
'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
```

PCS206　thread

概述

Python 是支持多线程的，并且是 native 的线程，主要是通过 thread 和 threading 这两个模块来实现的。thread 是比较底层的模块，threading 是 thread 的包装，可以更加方便地被使用。这里需要提一下的是 Python 对线程的支持还不够完善，不能利用多 CPU，但是下个版本的 Python 中已经考虑改进这点。我们不建议使用 thread 模块，是由于以下几点原因：

首先，更高级别的 threading 模块更为先进，对线程的支持更为完善，而且使用 thread 模块里的属性有可能会与 threading 冲突。

其次，低级别 thread 模块的同步原语很少（实际上只有一个），而 threading 模块则有很多。

还有一个不要使用 thread 的原因，就是它对你的进程什么时候应该结束完全没有控制，当主线程结束时，所有的线程都会被强制结束掉，没有警告，也不会有正常的清除工作。

应用

下面举个使用 thread 模块的简单例子：

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 import thread
5 from time import sleep, ctime
6
```

```
 7 def loop0():
 8       print 'start loop 0 at:', ctime()
 9       sleep(4)
10       print 'loop 0 done at:', ctime()
11
12 def loop1():
13       print 'start loop 1 at:', ctime()
14       sleep(2)
15       print 'loop 1 done at:', ctime()
16
17 def main():
18       print 'starting at:', ctime()
19       # 创建一个新线程，执行 loop0
20       thread.start_new_thread(loop0, ())
21       # 创建另一个新线程，执行 loop1
22       thread.start_new_thread(loop1, ())
23       sleep(6)
24       print 'all DONE at:', ctime()
25
26 if __name__ == '__main__':
27       main()
```

在这个例子中，主程序创建了两个线程后进入睡眠 6 秒，而两个线程分别执行 loop0 和 loop1，线程 0 在 loop0 中睡眠了 4 秒，而线程 1 在 loop1 中只睡眠了 2 秒，这样导致线程 1 先结束，线程 0 后结束，最后主程序 6 秒后退出。运行之后，可以看到如下结果：

```
~$ python pcs-206-1.py
starting at: Sun Aug 31 20:04:44 2008
start loop 0 at: Sun Aug 31 20:04:44 2008
start loop 1 at: Sun Aug 31 20:04:44 2008
loop 1 done at: Sun Aug 31 20:04:46 2008
loop 0 done at: Sun Aug 31 20:04:48 2008
all DONE at: Sun Aug 31 20:04:50 2008
```

PCS207　threading

概述

Python 是支持多线程的，并且是 native 的线程，主要是通过 thread 和 threading 这两个模块来实现的。thread 是比较底层的模块，threading 是 thread 的包装，可以更加方便地被使用。这里需要提一下的是 Python 对线程的支持还不够完善，不能利用多 CPU，但是下个版本的 Python 中已经考虑改进这点。

应用

threading 模块主要功能是让一些线程的操作对象化了，创建了叫 Thread 的类。

使用线程有两种方法，一种是创建线程要执行的函式，把这个函式传递进 Thread 对象里，让它来执行，如下所示。

```
 1 # -*- coding: utf-8 -*-
 2 import string, threading, time
 3
 4 def thread_main(a):
 5     global count, mutex
 6     # 获得线程名
 7     threadname = threading.currentThread().getName()
 8
 9     for x in xrange(0, int(a)):
10         # 取得锁
11         mutex.acquire()
12         count = count + 1
13         # 释放锁
```

```
    14        mutex.release()
    15        print threadname, x, count
    16        time.sleep(1)
    17
    18 def main(num):
    19    global count, mutex
    20    threads = []
    21
    22    count = 1
    23    # 创建一个锁
    24    mutex = threading.Lock()
    25    # 先创建线程对象
    26    for x in xrange(0, num):
    27        threads.append(threading.Thread(target=thread_main, args=(10,)))
    28    # 启动所有线程
    29    for t in threads:
    30        t.start()
    31    # 主线程中等待所有子线程退出
    32    for t in threads:
    33        t.join()
    34 if __name__ == '__main__':
    35    num = 4
    36    # 创建 4 个线程
    37    main(4)
```

在这个例子中，主程序创建了 4 个线程，每个线程对全局变量 count 进行加 1 运算。因为多个线程对同一个变量进行了加 1 操作，为了避免冲突，在修改变量 count 之前加锁，之后解锁释放。运行后，可以看到：

```
~$ python pcs-207-1.py
Thread-1 0 2
Thread-2 0 3
Thread-3 0 4
Thread-4 0 5
Thread-1 1 6
...
Thread-1 8 34
Thread-2 8 35
Thread-3 8 36
Thread-4 8 37
Thread-1 9 38
Thread-2 9 39
Thread-3 9 40
Thread-4 9 41
```

第二种方法是直接从 Thread 继承，创建一个新的类，把线程执行的代码放到这个新类里，如下所示。

```
# -*- coding: utf-8 -*-
import threading
import time

class Test(threading.Thread):
    def __init__(self, num):
        threading.Thread.__init__(self)
        self._run_num = num

    def run(self):
        global count, mutex
        threadname = threading.currentThread().getName()

        for x in xrange(0, int(self._run_num)):
            # 取得锁
            mutex.acquire()
            count = count + 1
            # 释放锁
            mutex.release()
            print threadname, x, count
            time.sleep(1)

if __name__ == '__main__':
    global count, mutex
    threads = []
    num = 4
    count = 1
    # 创建锁
    mutex = threading.Lock()
    # 创建线程对象
    for x in xrange(0, num):
        threads.append(Test(10))
    # 启动线程
    for t in threads:
        t.start()
    # 等待子线程结束
    for t in threads:
        t.join()
```

这段代码的运行结果和上面例子是相同的，只是创建线程的方法是直接继承了 threading.Thread 类，主程序中只要创建这个类的对象就可以了。

PCS208　dict4ini

特别说明

此节特邀 dict4Ini 模块的作者 Limodou 写的。Limodou 是中国 Python 早期用户之一，长期在业余时间钻研各种 Python 技术，进一步信息可以参考他的维基和博客。

维基地址：*http://wiki.woodpecker.org.cn/moin/LiModou*

精巧地址：*http://bit.ly/38y55d*

博客地址：*http://blog.donews.com/limodou* 或 *http://hi.baidu.com/limodou*

概述

为什么要开发 dict4Ini 呢？因为有时需要处理 ini 文件。Python 中的确已经存在像 ConfigParser 这样的模块，但是它只能处理标准的 ini 格式，处理的基本上都是字符串，如果要处理其他的格式要调用相应的转换函式，很不方便。在 Python 中使用得比较多的就是内置的 list、dict 之类的数据结构，许多人可能直接将这些数据写在.py 文件中，以.py 作为类 ini 文件来使用，但这样只能用在 Python 程序中，并且修改也不像 ini 文件那样方便。同时使用像 ConfigParser 这样的模块，不能非常方便地引用 ini 中的项。

仔细观察一下一个 ini 文件，它就像是一个 dict，并且可看成是两层的 dict。第一层就是 session，第二层就是 session 下的项。在使用时可以把它理解为 `ini['session']['key']`。

因此我就想如何将 ini 映射成为 dict 的形式，同时可以将 dict 保存为 ini 的形式呢？这两

者不就比较完美地统一了吗？因此我开发了 dict4Ini 这个模块。它最早是用在改进 confbot（一个 gtalk 的聊天机器人程序）上，后来慢慢发展，用在了像 UliPad 等多个项目中。

应用

dict4Ini 允许你把 ini 当成一个字典来使用，它支持传统的 ini 格式，即每项的值就是一个字符串。同时还支持非传统的使用，就是 Python 数据类型，可以支持数值、list、字符串格式，同时支持 unicode。因此在使用非传统格式时，它其实是把 Python 的数据表示写在了 ini 文件中，因此像字符串一般都有双引号，列表使用“,”进行分隔，数值可以直接写，包括浮点数，Unicode 字符串只要在字符串前面加 u 就可以了。

dict4Ini 除了支持简单的 Python 数据类型存储外，还可以：

- 对关键信息进行加密；
- 读取简单的注释，并且在保存时注释不会丢失；
- 记忆配置项的顺序，当写回文件时顺序不会乱；
- 实现分层的保存。

特别是在使用时，你把它当成一个 dict 就可以了，如：

```
x = dict4ini.DictIni('t.ini')
print x[“default”][“path”]
```

上面的代码将打开一个 t.ini 文件，同时将 default 节下的 path 项打印出来，那么这个 t.ini 的格式可以为：

```
[default]
path = "path"
```

dict4ini 不仅可以像字典一样操作，还可以像属性一样操作，前提是你的项的 key 应该符合一个标识符的定义，如上例可以改为：

```
print x.default.path
```

这样会更方便。并且如果你想创建一个新的项，直接赋值即可。如：

```
x.default.newvalue = 'value'
x.save()
```

这样就自动创建了一个新的项，然后调用 save()方法就可以保存起来了。

小结

dict4Ini 的项目主页在 *http://code.google.com/p/dict4ini/*上，上面有许多示例可以进行参考。另外还有一些注意事项：

（1）如果你想要保存 Python 数据类型，需要注意它的格式，它可能根据数据格式的不同而变化。如 list 会是以“,”分隔的串，如果字符串中有特殊符号，它将使用双引号进行处理。

（2）布尔类型需要处理为 1 或 0。

练习

1. 使用 dict4ini 创建一个新的 ini 文件，并包含如下内容：

```
 [default]
path = "hello,world"
name = name

[session]
file = testfile
```

2. 使用 dict4ini 创建一个新的 ini 文件，并包含如下内容：

```
 [default]
123 = "has spaces"
list = list,1,"hello, world",
```

PCS209　fnmatch

概述

fnmatch 实现了 shell patterns 表匹配字符串或文件名。

应用

fnmatch(name, pattern)

fnmatch(name, pattern)的函式功能：测试 name 是否匹配 pattern，匹配返回 true，否则返回 false。

```
>>>import fnmatch
# 匹配以.py 结尾的字符串
>>>fnmatch.fnmatch('*py','*.py')
False
>>>fnmatch.fnmatch('tlie.py','*.py')
True
```

注意：匹配样式是 unix shell 风格的。其中“*”表示匹配零个或多个字符。“?”表示匹配单个字符。“[seq]”匹配单个 seq 中的字符。“[!seq]”匹配单个不是 seq 中的字符。

filter(names, pat)

filter(names, pat)的函式功能：实现列表特殊字符的过滤或筛选，返回符合匹配模式的字符列表。

示例如下：

```
>>>import fnmatch
>>>names = ['dlsf', 'ewro.txt', 'te.py', 'youe.py']
# 匹配以.py 结尾的字符串
>>> fnmatch.filter(names, '*.py')
['te.py', 'youe.py']
>>> fnmatch.filter(names, '[de]')
[]
# 匹配以 d 或 e 开头的字符串
>>>fnmatch.filter(names, '[de]*')
['dlsf', 'ewro.txt']
```

fnmatchcase(name, pat)

fnmatchcase(name, pat)的函式功能：和 fnmatch()类似，它只是 fnmatchcase 强制区分大小写匹配，不管文件系统是否区分。

示例如下：

```
>>> import fnmatch
# 匹配以 R 开头的字符串
>>> fnmatch.fnmatchcase('readme', 'R*')
False
>>> fnmatch.fnmatchcase('Readme', 'R*')
True
```

translate(pat)

translate(pat)的函式功能：将 pat 转换成正则表达式。

示例如下：

```
>>> import fnmatch
>>> fnmatch.translate('*.py')
'.*\\.py$'
```

PCS210　pickle

概述

pickle 模块可以实现任意的 Python 对象的序列化。这些序列化后的对象可以被传输或存储，也可以重构为一个和原先对象具有相同特征的新对象。但在实际项目中为了提高序列化和反序列化的速度，可以改用 cPickle 模块。pickle 的文档清晰地表明它不提供安全保证。所以慎用 pickle 来作为内部进程通信或者数据存储，也不要相信那些你不能验证安全性的数据。

应用

第一个 pickle 示例是将一个数据结构编码为一个字符串，然后将其输出到控制台，如下所示。

```
1 try:
2   import cPickle as pickle
3 except:
4   import pickle
5 import pprint
```

首先尝试导入 cPickle，并指定别名为“pickle”。如果因为某种原因导入 pickle 失败，则导入由 Python 实现的 pickle 模块。这样，如果 cPickle 可用则速度会快点，但如果不可用，也会有正确的实现。接下来，定义一个完全由基本类型组成的数据结构，如下所示。

```
1 data = [ { 'a':'A', 'b':2, 'c':3.0 } ]
```

```
2 print 'DATA:',
3 pprint.pprint(data)
```

可以使用 pickle.dumps()来创建数据值的字符串表示，如下所示。

```
1 data_string = pickle.dumps(data)
2 print 'PICKLE:', data_string
```

可以看到结果如下：

```
~$ python pcs-201-1.py
DATA:[{'a': 'A', 'b': 2, 'c': 3.0}]
PICKLE: (lp1
(dp2
S'a'
S'A'
sS'c'
F3
sS'b'
I2
sa.
```

一旦数据被序列化，就可以把他写入到文件、socket、管道等中。之后你可以读取这个文件，读取这些数据来构造具有相同值的新对象，如下所示。

```
 1 # -*- coding: utf-8 -*-
 2 data1 = [ { 'a':'A', 'b':2, 'c':3.0 } ]
 3 print 'BEFORE:',
 4 pprint.pprint(data1)
 5
 6 data1_string = pickle.dumps(data1)
 7
 8 data2 = pickle.loads(data1_string)
 9 print 'AFTER:',
10 pprint.pprint(data2)
11
12 print 'SAME?:', (data1 is data2)
13 print 'EQUAL?:', (data1 == data2)
```

新构造的对象等于原来的对象，但他们又不是相同的对象。

```
~$ python pcs-201-2.py
BEFORE:[{'a': 'A', 'b': 2, 'c': 3.0}]
AFTER:[{'a': 'A', 'b': 2, 'c': 3.0}]
SAME?: False
EQUAL?: True
```

在处理自定义类时，应该保证这些被序列化的类会在进程名空间出现 。只有数据实例才

能被序列化，而不能是定义的类。在反序列化时，类的名字被用于寻找构造器以便创建新对象。接下来这个例子，是将一个类实例写入到文件中：

```
# -*- coding: utf-8 -*-
try:
  import cPickle as pickle
except:
  import pickle
import sys

class SimpleObject(object):

  def __init__(self, name):
      self.name = name
      l = list(name)
      l.reverse()
      self.name_backwards = ''.join(l)
      return

if __name__ == '__main__':
  data = []
  data.append(SimpleObject('pickle'))
  data.append(SimpleObject('cPickle'))
  data.append(SimpleObject('last'))

  try:
      filename = sys.argv[1]
  except IndexError:
      raise RuntimeError('Please specify a filename as an argument to %s'
       % sys.argv[0])

  out_s = open(filename, 'wb')
  try:
      # 写入到流中
      for o in data:
         print 'WRITING: %s (%s)' % (o.name, o.name_backwards)
         pickle.dump(o, out_s)
  finally:
      out_s.close()
```

当运行这个脚本时，它会创建名为命令行中输入参数的文件，如下所示：

```
~$ python pcs-201-3.py test.dat
WRITING: pickle (elkcip)
```

```
WRITING: cPickle (elkciPc)
WRITING: last (tsal)
```

一个简单的尝试是将刚才的序列化对象装载进来，如下所示：

```
 1 # -*- coding: utf-8 -*-
 2 try:
 3   import cPickle as pickle
 4 except:
 5   import pickle
 6 import pprint
 7 from StringIO import StringIO
 8 import sys
 9 #注意:需要导入自定义类
10 from pcs-201-3 import SimpleObject
11
12 try:
13   filename = sys.argv[1]
14 except IndexError:
15   raise RuntimeError('Please specify a filename as an argument to %s'
      % sys.argv[0])
16
17 in_s = open(filename, 'rb')
18 try:
19   # 读取数据
20   while True:
21      try:
22         o = pickle.load(in_s)
23      except EOFError:
24        break
25      else:
26         print 'READ: %s (%s)' % (o.name, o.name_backwards)
27 finally:
28   in_s.close()
```

运行这个脚本，可以看到：

```
~$ python pcs-201-4.py test.dat
READ: pickle (elkcip)
READ: cPickle (elkciPc)
READ: last (tsal)
```

PCS211　base64

概述

Base64 是网络上最常见的用于传输 8 bit 字节代码的编码方式之一，大家可以查看 RFC2045～RFC2049，上面有 MIME 的详细规范。Base64 要求把每三个 8 bit 的字节转换为四个 6bit 的字节（3×8 = 4×6 = 24），然后把 6 bit 的字节再添两位高位 0，组成四个 8 bit 的字节，也就是说，转换后的字符串理论上将要比原来的长 1/3。当然，Python 的 base64 库，帮我们封装了这些复杂的算法，只须简单地使用 base64.encodestring("Hello Python")，就可以进行 base64 编码了。

应用

Base 64 编码

简单的文本编码示例如下：

```
1
2 import base64
3
4 initial_data = open(__file__, 'rt').read()
5
6 encoded_data = base64.b64encode(initial_data)
```

```
 7
 8 num_initial = len(initial_data)
 9 padding = { 0:0, 1:2, 2:1 }[num_initial % 3]
10
11 print '%d bytes before encoding' % num_initial
12 print 'Expect %d padding bytes' % padding
13 print '%d bytes after encoding' % len(encoded_data)
14 print
15 print encoded_data
```

输出显示原来 350 字节的文本在编码之后被扩展到了 468 个字节（如下所示），从编码过程来看，每一个 24 位序列（3 个字节）作为输入，输出时候则增加为 4 个字节，最后一个字符为“=”，它是简单的追加，因为原始字符串的位数不能被 24 整除。

```
~$ python pcs-211-1.py
350 bytes before encoding
Expect 1 padding bytes
468 bytes after encoding

aW1wb3J0IGJhc2U2NAoKaW5pdGlhbF9kYXRhID0gb3BlbihfX2ZpbGVfXywgJ3J0JykucmV
hZCgpCgplbmNvZGVkX2
RhdGEgPSBiYXNlNjQuYjY0ZW5jb2RlKGluaXRpYWxfZGF0YSkKCm51bV9pbml0aWFsID0gb
GVuKGluaXRpYWxfZG
F0YSkKcGFkZGluZyA9IHsgMDowLCAxOjIsIDI6MSB9W251bV9pbml0aWFsICUgM10KCnBya
W50ICclZCBieXRlcyBi
ZWZvcmUgZW5jb2RpbmcnICUgbnVtX2luaXRpYWwKcHJpbnQgJ0V4cGVjdCAlZCBwYWRkaW5
nIGJ5dGVzJyAlIHB
hZGRpbmcKcHJpbnQgJyVkIGJ5dGVzIGFmdGVyIGVuY29kaW5nJyAlIGxlbihlbmNvZGVkX2
RhdGEpCnByaW50CnBy
aW50IGVuY29kZWRfZGF0YQ=
```

Base 64 解码

编码的字符串可以转换为原来的格式，利用反向转换把 4 个字节转换为 3 个字节，可以使用 b64decode()函式，如下所示。

```
 1 import base64
 2
 3 original_string = 'This is the data, in the clear.'
 4 print 'Original:', original_string
 5
 6 encoded_string = base64.b64encode(original_string)
```

```
 7 print 'Encoded :', encoded_string
 8
 9 decoded_string = base64.b64decode(encoded_string)
10 print 'Decoded :', decoded_string
```

输出显示为：

```
~$ python pcs-211-2.py
Original: This is the data, in the clear.
Encoded : VGhpcyBpcyB0aGUgZGF0YSwgaW4gdGhlIGNsZWFyLg==
Decoded : This is the data, in the clear.
```

URL-Safe 变体

默认的 base64 字母表可能会使用+和/，而这些字符可能出现在 URL 中，因此必须为这些字符指定可选择的编码情况，+由符号–来代替，而/由下画线_来代替，其他字母表不变，如下所示。

```
 1 import base64
 2
 3 for original in [ '\xfb\xef', '\xff\xff' ]:
 4    print 'Original       :', repr(original)
 5    print 'Standard encoding:', base64.standard_b64encode(original)
 6    print 'URL-safe encoding:', base64.urlsafe_b64encode(original)
 7    print
```

输出显示为：

```
~$ python pcs-211-3.py
Original       : '\xfb\xef'
Standard encoding: ++8=
URL-safe encoding: --8=

Original       : '\xff\xff'
Standard encoding: //8=
URL-safe encoding: __8=
```

其他编码

除了 base 64 以外，还有 base 32 和 base 16（16 进制）提供函式，用于编码数据，如下所示。

```
 1 import base64
 2
 3 original_string = 'This is the data, in the clear.'
 4 print 'Original:', original_string
 5
 6 encoded_string = base64.b32encode(original_string)
 7 print 'Encoded :', encoded_string
 8
 9 decoded_string = base64.b32decode(encoded_string)
10 print 'Decoded :', decoded_string
```

输出显示为：

```
~$ python pcs-211-4.py
Original: This is the data, in the clear.
Encoded : KRUGS4ZANFZSA5DIMUQGIYLUMEWCA2LOEB2GQZJAMNWGKYLSFY======
Decoded : This is the data, in the clear.
```

base 16 中的函式是以十六进制方式表示的，如下所示。

```
 1 import base64
 2
 3 original_string = 'This is the data, in the clear.'
 4 print 'Original:', original_string
 5
 6 encoded_string = base64.b16encode(original_string)
 7 print 'Encoded :', encoded_string
 8
 9 decoded_string = base64.b16decode(encoded_string)
10 print 'Decoded :', decoded_string
```

输出显示为：

```
~$ python pcs-211-5.py
Original: This is the data, in the clear.
Encoded : 546869732069732074686520646174612C20696E2074686520636C6561722E
Decoded : This is the data, in the clear.
```

PCS212　shutil

概述

shutil 模块提供了一些高层次的文件操作，比如复制、设置权限、删除等。

应用

copyfile()

copyfile()将源文件内容完全复制给目标文件。如果没有写入目标文件的权限，会引起 IOError。另外，对于一些特殊文件，使用 copyfile()是不能被复制成新的特殊文件的，如下所示。

```
1 import os
2 from shutil import *
3
4 print 'BEFORE:', os.listdir(os.getcwd())
5 copyfile('pcs-212-1.py', 'pcs-212-1.py.copy')
6 print 'AFTER:', os.listdir(os.getcwd())
```

输出显示为：

```
~$ python pcs-212-1.py
BEFORE: ['pcs-212-1.py', 'pcs-212.moin']
AFTER: ['pcs-212-1.py.copy', 'pcs-212-1.py', 'pcs-212.moin']
```

copyfile()其实是底层调用了 copyfileobj()函式。文件名参数传递给 copyfile()后，进而将此文件句柄传递给 copyfileobj()，并由它真正完成文件复制。

copy()

copy()函式类似于 Unix 命令 cp，将源文件复制成一个目标文件。和 copyfile()不同的是，如果目标参数是一个目录而不是文件，那么在这个目录中复制一个源文件副本（它与源文件同名）。另外，文件的权限也会随之被复制，如下所示。

```
1 import os
2 from shutil import *
3
4 os.mkdir('example')
5 print 'BEFORE:', os.listdir('example')
6 copy('pcs-212-2.py', 'example')
7 print 'AFTER:', os.listdir('example')
```

运行结果为：

```
~$ python pcs-212-2.py
BEFORE: []
AFTER: ['pcs-212-2.py']
```

同时还会在当前目录下面创建一个 example 目录，并把 pcs-212-2.py 文件 copy 到 example 目录里面。

copy2()

copy2()函式类似于 copy()，但不同于 copy()的地方是，copy2()将一些元信息，如文件最后一次被读取时间和修改时间等，也复制给了新文件，如下所示。

```
 1 import os
 2 import time
 3 from shutil import *
 4
 5 def show_file_info(filename):
 6     stat_info = os.stat(filename)
 7     print '\tMode    :', stat_info.st_mode
 8     print '\tCreated :', time.ctime(stat_info.st_ctime)
 9     print '\tAccessed:', time.ctime(stat_info.st_atime)
10     print '\tModified:', time.ctime(stat_info.st_mtime)
```

```
11
12 os.mkdir('example')
13 print 'SOURCE:'
14 show_file_info('pcs-212-3.py')
15 copy2('pcs-212-3.py', 'example')
16 print 'DEST:'
17 show_file_info('example/pcs-212-3.py')
```

运行结果如下，可以看到新文件的元信息和源文件是一样的：

```
~$ python pcs-212-3.py
SOURCE:
   Mode   : 33188
   Created : Sun Sep 14 16:49:31 2008
   Accessed: Sun Sep 14 16:49:33 2008
   Modified: Sun Sep 14 16:49:31 2008
DEST:
   Mode   : 33188
   Created : Sun Sep 14 16:49:39 2008
   Accessed: Sun Sep 14 16:49:33 2008
   Modified: Sun Sep 14 16:49:31 2008
```

copymode()

在 Unix 下，一个新创建文件的权限会根据当前用户的 umask 值来设置。这里可以使用 copymode()来实现使被创建的文件具有 umask 值，而不是由于源文件的权限，如下所示。

```
1 from commands import *
2 from shutil import *
3
4 print 'BEFORE:', getstatus('file_to_change.txt')
5 copymode('pcs-212-4.py', 'file_to_change.txt')
6 print 'AFTER :', getstatus('file_to_change.txt')
```

首先创建一个文件 file_to_change.txt。然后对其权限做些修改：

```
~$ touch file_to_change.txt
~$ chmod ugo+w file_to_change.txt
```

接着，运行刚才的示例脚本：

```
~$ python pcs-212-4.py
BEFORE: -rw-rw-rw- 1 shengyan shengyan 0 2008-09-14 16:54 file_to_change.txt
AFTER : -rw-r--r-- 1 shengyan shengyan 0 2008-09-14 16:54 file_to_change.txt
```

可以看到两个文件的权限是不一样的，新的 file_to_change.txt 的权限设置是与普通新文件

的权限一致的。

copystat()

copystat()函式可以复制文件的其他元信息（权限、最后读取时间、最后修改时间等）。它和 copy2()函式很类似。

```
 1 import os
 2 from shutil import *
 3 import time
 4
 5 def show_file_info(filename):
 6    stat_info = os.stat(filename)
 7    print '\tMode   :', stat_info.st_mode
 8    print '\tCreated :', time.ctime(stat_info.st_ctime)
 9    print '\tAccessed:', time.ctime(stat_info.st_atime)
10    print '\tModified:', time.ctime(stat_info.st_mtime)
11
12 print 'BEFORE:'
13 show_file_info('file_to_change.txt')
14 copystat('pcs-212-5.py', 'file_to_change.txt')
15 print 'AFTER :'
16 show_file_info('file_to_change.txt')
```

运行结果如下：

```
~$ python pcs-212-5.py
BEFORE:
   Mode   : 33188
   Created : Sun Sep 14 16:54:35 2008
   Accessed: Sun Sep 14 16:54:36 2008
   Modified: Sun Sep 14 16:54:09 2008
AFTER :
   Mode   : 33188
   Created : Sun Sep 14 16:58:44 2008
   Accessed: Sun Sep 14 16:58:44 2008
   Modified: Sun Sep 14 16:58:42 2008
```

copytree()

使用 copytree()来复制目录，它会递归复制整个目录结构。目标目录必须不存在。其中的

symlinks 参数控制符号链接是否作为链接或文件被复制，默认的是将其内容复制成一个新文件。如果此选项为 true，新的链接会在目标目录中创建，如下所示。

```
1 from commands import *
2 from shutil import *
3
4 print 'BEFORE:'
5 print getoutput('ls -rlast ./example_other')
6 copytree('example', './example_other')
7 print 'AFTER:'
8 print getoutput('ls -rlast ./example_other')
```

运行结果为：

```
~$ python pcs-212-6.py
BEFORE:
ls: 无法访问./example_other: 没有该文件或目录
AFTER:
总用量 12
4 -rw-r--r-- 1 shengyan shengyan  473 2008-09-14 16:49 pcs-212-3.py
4 drwxr-xr-x 2 shengyan shengyan 4096 2008-09-14 16:49 .
4 drwxr-xr-x 5 shengyan shengyan 4096 2008-09-14 17:04 ..
```

rmtree()

使用 rmtree()可以删除整个目录。若其中产生错误，会作为异常抛出。但是如果它的第二个参数是目录，那么产生的错误会被忽略，第三个参数可以指定为一个特殊出错处理函式句柄，如下所示。

```
1 from commands import *
2 from shutil import *
3
4 print 'BEFORE:'
5 print getoutput('ls -rlast ./example_other')
6 rmtree('example_other', './example_other')
7 print 'AFTER:'
8 print getoutput('ls -rlast ./example_other')
```

运行结果为：

```
~$ python pcs-212-7.py
BEFORE:
总用量 12
4 -rw-r--r-- 1 shengyan shengyan  473 2008-09-14 16:49 pcs-212-3.py
4 drwxr-xr-x 2 shengyan shengyan 4096 2008-09-14 16:49 .
```

```
4 drwxr-xr-x 5 shengyan shengyan 4096 2008-09-14 17:04 ..
AFTER:
ls: 无法访问./example_other: 没有该文件或目录
```

move()

移动文件或目录可以使用 move()，这类似于 Unix 命令 mv。如果源文件或目录和目标文件或目录在同一个文件系统下，那么源文件或目录会直接重命名，否则源文件或目录会复制到目标文件或目录，接着删除源文件或目录，如下所示。

```
1 import os
2 from shutil import *
3
4 print 'BEFORE: example : ', os.listdir('example')
5 move('example', 'example2')
6 print 'AFTER : example2: ', os.listdir('example2')
```

运行结果为：

```
~$ python pcs-212-8.py
BEFORE: example :  ['pcs-212-3.py']
AFTER : example2:  ['pcs-212-3.py']
```

PCS213　time

概述

time 是 Python 的时间处理模块。时间处理是在编程的时候经常遇到的一个问题，经常遇到的问题有：如何获取当前的 Unix 时间戳？怎样求出 15 分钟后是几点几分？如何格式化时间？如何设置时区信息等。

应用

time.time()

可以直接使用 time.time()获取当前时间戳，如：

```
1 import time
2 print 'The time is:', time.time()
```

运行结果：

```
~$ python pcs-213-1.py
The time is: 1209465739.3
```

time.ctime()

当存储和比较日期时，浮点型一般是很有用的，但这种方式不易阅读，为了获取更易读

的记录和输出时间值可以使用 time.ctime()：

```
1 import time
2 print 'The time is     :', time.ctime()
3 later = time.time() + 15
4 print '15 secs from now :', time.ctime(later)
```

上例表明了如何利用 ctime()函式对当前时间进行格式化，运行后可以看到：

```
~$ python  pcs-213-2.py
The time is     : Sun Mar  9 12:18:02 2008
15 secs from now : Sun Mar  9 12:18:17 2008
```

time.clock()

time()函式返回的是现实世界的时间，而 clock()函式返回的是 cpu 时钟。clock()函式返回值常用作性能测试、benchmarking 等。它们反映的是程序运行的真实时间，比 time()函式返回的值要精确。

```
 1 # -*- coding: utf-8 -*-
 2 import hashlib
 3 import time
 4
 5 # 用于计算 md5 校验和的数据
 6
 7 data = open(__file__, 'rt').read()
 8
 9 for i in range(5):
10     h = hashlib.sha1()
11     print time.ctime(), ': %0.3f %0.3f' % (time.time(), time.clock())
12     for i in range(100000):
13         h.update(data)
14     cksum = h.digest()
```

在这个例子中，ctime()把 time()函式返回的浮点型表示转化为标准时间，每个迭代循环使用了 clock()。如果想在机器上测试这个例子，那么可以增加循环次数，或者处理大一点的数据，这样才能看到不同点。

```
~$ python pcs-213-3.py
Sun Sep  7 16:05:44 2008 : 1220774744.580 0.010
Sun Sep  7 16:05:44 2008 : 1220774744.891 0.240
Sun Sep  7 16:05:45 2008 : 1220774745.221 0.490
Sun Sep  7 16:05:45 2008 : 1220774745.546 0.720
Sun Sep  7 16:05:45 2008 : 1220774745.861 0.970
```

但需要注意的是，如果程序没有做任何事情，处理器时钟是不会计时的。

struct_time

有时候你需要获取日期的单独部分（如年、月等），time 模块定义了 struct_time 来存储日期和时间值并作为它的一部分以便获取。并提供了多种函式将 struct_time 转化为 float。

```
 1 import time
 2
 3 print 'gmtime   :', time.gmtime()
 4 print 'localtime:', time.localtime()
 5 print 'mktime   :', time.mktime(time.localtime())
 6
 7 print
 8 t = time.localtime()
 9 print 'Day of month:', t.tm_mday
10 print ' Day of week:', t.tm_wday
11 print ' Day of year:', t.tm_yday
```

gmtime()返回当前的 UTC 时间，localtime()返回当前时间域的当前时间，mktime()接收 struct_time 参数并将其转化为浮点型来表示。

```
~$ python pcs-213-4.py
gmtime   : (2008, 9, 7, 8, 8, 13, 6, 251, 0)
localtime: (2008, 9, 7, 16, 8, 13, 6, 251, 0)
mktime   : 1220774893.0

Day of month: 7
Day of week: 6
Day of year: 251
```

解析和格式化时间

函式 strptime()和 strftime()可以使 struct_time 和时间值字符串相互转化。下面示例把当前时间（字符串）转化为 struct_time 实例，然后再转化为字符串。

```
 1 import time
 2
 3 now = time.ctime()
 4 print now
 5 parsed = time.strptime(now)
```

```
 6 print parsed
 7 print time.strftime("%a %b %d %H:%M:%S %Y", parsed)
```

输出和输入字符串不是完全的一致，主要表现在月份前加了一个 0 前缀。

```
~$ python pcs-213-5.py
Sun Sep  7 16:10:09 2008
(2008, 9, 7, 16, 10, 9, 6, 251, -1)
Sun Sep 07 16:10:09 2008
```

使用 Time Zone（时区）

无论是你的程序，还是为系统使用一个默认的时区，检测当前时间的函式依赖于当前时区的设置。改变时区设置不会改变实际时间，只会改变表示时间的方法。

通过设置环境变量 TZ 可以改变时区，然后调用 tzset()。环境变量 TZ 可以对时区作详细的设置，比如白天保存时间的起始点。下面这个示例改变了 time zone 中的一些值，展示了这种改变如何来影响 time 模块中的其他设置。

```
 1 import time
 2 import os
 3
 4 def show_zone_info():
 5     print '\tTZ    :', os.environ.get('TZ', '(not set)')
 6     print '\ttzname:', time.tzname
 7     print '\tZone  : %d (%d)' % (time.timezone, (time.timezone / 3600))
 8     print '\tDST   :', time.daylight
 9     print '\tTime  :', time.ctime()
10     print
11
12 print 'Default :'
13 show_zone_info()
14
15 for zone in [ 'US/Eastern', 'US/Pacific', 'GMT']:
16     os.environ['TZ'] = zone
17     time.tzset()
18     print zone, ':'
19     show_zone_info()
```

```
~$ python pcs-213-6.py
Default :
   TZ    : (not set)
```

```
    tzname: ('CST', 'CST')
    Zone  : -28800 (-8)
    DST   : 0
    Time  : Sun Sep  7 16:12:48 2008

US/Eastern :
    TZ    : US/Eastern
    tzname: ('EST', 'EDT')
    Zone  : 18000 (5)
    DST   : 1
    Time  : Sun Sep  7 04:12:48 2008

US/Pacific :
    TZ    : US/Pacific
    tzname: ('PST', 'PDT')
    Zone  : 28800 (8)
    DST   : 1
    Time  : Sun Sep  7 01:12:48 2008

GMT :
    TZ    : GMT
    tzname: ('GMT', 'GMT')
    Zone  : 0 (0)
    DST   : 0
    Time  : Sun Sep  7 08:12:48 2008
```

PCS214　ElementTree

概述

ElementTree 是 Python 的 XML 处理模块，它提供了一个轻量级的对象模型。它在 Python 2.5 以后成为 Python 标准库的一部分，但在 Python 2.4 之前需要单独安装。

应用

先看一个最简单的例子。

```
 1 # -*- coding: utf-8 -*-
 2 import xml.etree.ElementTree as ET
 3
 4 # 创建一个根节点 root,并设置其标签为"html"
 5 root = ET.Element("html")
 6 # 创建根节点的一个子节点 head,并设置其标签为"head"
 7 head = ET.SubElement(root, "head")
 8 # 创建节点 head 的一个子节点 title,并设置其标签为"title",设置其内容为"Title"
 9 title = ET.SubElement(head, "title")
10 title.text = "Title"
11
12 # 创建根节点的一个子节点 body,并设置其标签为"body",设置 bgcolor 属性为#ffffff,
     设置其内容为"Hello, World!"
13 body = ET.SubElement(root, "body")
14 body.set("bgcolor", "#ffffff")
```

```
15 body.text = "Hello, World!"
16
17 # 将这个根节点包裹在 ElementTree 对象中,并且保存为 XML 格式文档
18 tree = ET.ElementTree(root)
19 tree.write("hello.xhtml")
```

使用 Element 新建 XML 节点，并设置相应值，然后将其封装在 ElementTree 中，构成最终 XML 树保存到本地文件中。运行这个程序，可以看到在当前目录下生成了一个 hello.xhtml 文件，用 firefox 或其他浏览器打开，可以看到图 PCS214-1：

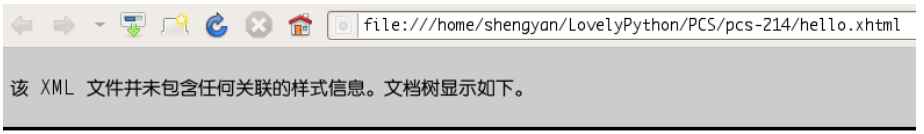


图 PCS214-1

使用 ElementTree 的 parser()函式可以快速加载一个完整的 XML 文档并生成一个 ElementTree 对象，例如：

```
1 # -*- coding: utf-8 -*-
2 import xml.etree.ElementTree as ET
3
4 # 使用 parse()将 XML 文档加载并返回 ElementTree 对象
5 tree = ET.parse("hello.xhtml")
6
7 # 寻找 head 节点下 title 节点的值
8 print tree.findtext("head/title")
9
10 # 使用 getroot()函式返回根节点
11 root = tree.getroot()
12
13 # ...接下来可以做些其他操作
14
15 # 保存为本地 XML 文档
16 tree.write("out.xml")
```

ElementTree 还提供了很多类和函式。类有——

- ElementTree：经常使用的元素结构封装类。
- TreeBuilder：普通元素结构构造器。

- XMLTreeBuilder：对 XML 源数据构造成 XML 结构树，它是基于 expat 解析器实现的。

函式有——

- XML(text)：从一个包含 XML 数据的字串文本中解析出 XML 元素。
- dump(elem)：将整个元素结构树输出至标准输出 sys.stdout。
- iselement(element)：判断参数元素是否为有效的元素对象。

还有很多，可以参见网址：*http://effbot.org/zone/pythondoc-elementtree-ElementTree.htm*

精巧地址：*http://bit.ly/3PbG6Q*

PCS215　random

概述

生活中，处处充满着随机事件和数据。在丰富多彩的 Python 世界中，random 模块用于生成随机数、随机字符和随机字符串等。

应用

random.randint()

random.randint()的函式功能：生成随机整数。

```
1 # coding:utf-8
2 import random
3 #打印随机生成的整数（1～9）
4
5 print '随机生成 1～9 的整数：%d' %random.randint(1, 9)
```

可以通过参数限定生成随机整数的范围。

运行结果：

```
~$ python pcs-215-1.py
随机生成 1～9 的整数：6
```

random.randrange()

random.randrange()的函式功能：随机选取指定整数序列中的某个元素。

```
1 # coding:utf-8
2 import random
3 #打印随机生成的偶数（20～200）
4
5 print '随机生成 20～200 的偶数：%d' %random.randrange(20, 201, 2)
```

改变第三个参数便可以更加灵活地生成想要的随机数。如当第三个参数为 10 时，生成的随机数便是 10 的倍数。

运行结果：

```
~$ python pcs-215-2.py
随机生成 20～200 的偶数：198
```

random.random()和 random.uniform()

random.random()和 random.uniform()的函式功能：生成随机浮点数。

```
1 # coding:utf-8
2 import random
3 #打印随机生成的浮点数
4
5 print '随机生成 0～1 的浮点数：%f' %random.random()
6 print '随机生成 1～20 的浮点数：%f' %random.uniform(1, 20)
```

注意，random.random()不能传递参数，它只能生成 0～1 的浮点数。相比直线 random.uniform()方法更加灵活。

运行结果：

```
~$ python pcs-215-3.py
随机生成 0～1 的浮点数：0.259886
随机生成 1～20 的浮点数：17.918067
```

random.choice()和 random.sample()

random.choice()和 random.sample()的函式功能：生成随机字符、字符串。

```
1 # coding:utf-8
2 import random, string
```

```
3 #打印随机生成的字符、字符串
4
5 print '随机生成的字符(a~z)：%c' %random.choice('abcdefghijklmnopqrstuvwxyz')
6 print '随机生成的字符串(春、夏、秋、冬):%s' %random.choice(['spring', 'summer',
 'fall', 'winter'])
7 print '随机生成的字符串：%s' %string.join(random.sample
  ('abcdefghijklmnopqrstuvwxyz', 4), '')
```

random.sample()方法返回的是一个列表，这里使用 string.join()方法将其转化成字符串。

运行结果：

```
~$ python pcs-215-4.py
随机生成的字符(a~z)：p
随机生成的字符串(春、夏、秋、冬)：winter
随机生成的字符串：qudt
```

random.shuffle()

random.shuffle()的函式功能：打乱排序。

```
1 # coding:utf-8
2 import random
3 #打印随机排序结果
4
5 items = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
6 print '%s' %items
7 random.shuffle(items)
8 print '随机排序结果为：\n%s' %items
```

注意：调用 random.shuffle()方法直接修改了 items 的值，返回 None。

运行结果：

```
~$ python pcs-215-5.py
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

随机排序结果为：

```
[2, 1, 8, 9, 4, 3, 5, 6, 7, 0]
```

提醒：

1. 在生成字符串时，要根据具体的要求选择合适的方法，同时要注意程序的效率和代码的可读性。其实单个字符就是非常特殊的字符串。
2. 生成随机值时，注意传递限制范围的参数是否包含边界处的值。

PCS216　socket

概述

socket 的英文意思是“孔、插座”，在这里它是一种进程的通信机制。可以把 socket 比作电话插座，区号是通话双方的网络地址；一方电话机发出信号和另一方接受信号，相当于向 socket 发送数据和从 socket 接收数据；通话结束后挂起电话机，相当于关闭 socket，撤销链接。

Python 中提供了丰富的网络编程模块，适当地调用这些模块可以大大提高编程效率，而且能够保证程序的健壮性。同时你会惊奇发现，Python 作为脚本语言，对部分网络协议的实现相当快。这里介绍的 socket 模块，是 Python 网络编程最基础的一个模块。当然在进行网络编程之前你必须清楚网络基本体系结构和基本原理，如果你做过类似的 C 语言网络编程，你会发现 Python 网络编程的流程与之非常相似。

应用

建立网络服务端

网络服务端的建立过程分为 6 个阶段，创建 Socket 对象、绑定端口、监听连接、接受请求、数据收发、关闭端口。

分别对应函式　socket.socket()、socket.bind()、socket.listen()、socket.accept()、

socket.sendall()\socket.recv()、socket.close()。

```
# coding:utf-8
import socket
# 监听本机 5678 端口，当有客户端请求时，回复指定字符串

host = '127.0.0.1 '
port = 5678

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#采用 TCP 协议
s.bind((host, port))
#绑定 socket
s.listen(1)
#服务器监听连接

while 1:
#这里的 whil 循环，可以使程序多地响应多客户端的请求
    clientSocket,clientAddr = s.accept()
    #接受客户端连接
    print 'Client connected!'
    clientSocket.sendall('Welcome to Python World!')
    #向客户端发送字符串
    clientSocket.close()
    #关闭与客户端的连接
```

这里采用的是可靠连接 TCP 协议，与之相对应的是不可靠的无连接的 UDP 协议。创建 socket 对象的协议类型，由 socket.socket()的第二个参数决定。socket.SOCK_STREAM 代表 TCP 协议，socket.SOCK_DGRAM 代表 UDP 协议。当然采用不同协议的 socket 对象的操作函式有些差异，这里列举的两个例子都是采用 TCP 协议的。

例子中选取的端口为 5678，系统中的端口分为两类：0～1023 为周知端口，一般供特定网络服务使用，1024～65535 为动态端口，供一般应用程序使用。

运行结果：

```
$ python pcs-216-1.py
Client connected!
```

建立网络客户端

网络服务端的建立过程分为四个阶段，创建 Socket 对象、连接服务器、数据收发、关闭

端口。

分别对应函式 socket.socket()、socket.connect()、socket.sendall()\socket.recv()、socket.close()

```
 1 # coding:utf-8
 2 import socket
 3 # 连接本机 5678 端口，并读取 socket，将结果打印屏幕
 4
 5 host = '127.0.0.1'
 6 port = 5678
 7
 8 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
 9 #创建采用 TCP 协议的 socket 对象
10 s.connect((host, port))
11 #连接指定的服务器端
12
13 while 1:
14 #从 socket 读取数据，并输出到屏幕，读完后退出程序
15     buf = s.recv(2048)
16     if not len(buf):
17         break
18     print buf
```

运行结果：

```
~$ python pcs-216-2.py
Welcome to Python World!
```

Socket 异常

Python 的 socket 模块定义了四种可能出现的异常。

1. socket.error：与一般 I/O 操作和数据通信问题有关的异常。
2. socket.gaierror：与域名解析有关的异常。
3. socket.herror：与其他地址解析错误有关的异常。
4. socket.timeout：socket 调用 sockettimeout()函式后，处理超时有关的异常。

关于异常处理请见语法篇中异常处理相关章节。

Socket 编程问题

1. 网络编程过程中的异常处理很重要，既要保证数据的准确性，也不能影响到数据处理的速度。异常处理函式，要有效利用异常信息，和子过程返回信息，准确定位错误，尽可能地解决错误继续程序的运行。如果不行，也得输出或记录详细有用的错误信息。
2. Python 提供了很多与协议相关的模块，如 ftplib、telnetlib、pop 等，利用这些库里提供的丰富函式，可以避开具体协议的实现，提高编码效率。Python 也提供了访问底层操作系统 Socket 接口的全部方法，这些接口可以为你提供灵活而强大的功能。当然有时你不得不自己去实现一些协议，可能是处理效率，可能是 Python 为之提供的相应模块。这里（*http://code.google.com/p/spyftp*、精巧地址：*http://bit.ly/2uucY5*）有一个用 Python 编写的 FTP（File Transfer Protocol）客户端，里边就是调用了 ftplib 库。实现所有的命令才只有 300 行代码，不过目前异常处理还不够，如果你愿意也可以帮忙去完善它。
3. Python 提供了一个 makefile()函式，可以用它来声称供编程人员使用的文件对象，这样对 socket 的读写就更加方便了。readlines()、write()、read()等函式都可以正常使用。

深入 Python 网络编程

考虑本书面向 Python 入门用户，这里所介绍的只是 Python 网络编程的冰山一角。如果你对网络编程感兴趣，那么 Python 的网络编程很值得你去深入学习。其灵活多变、功能强大不亚于 Perl 脚本语言。推荐一本 John Goerzen 写的《Foundations of Python Network Programming》（中文名《Python 网络编程基础》）可供参考。

PCS217　Tkinter

概述

Tkinter 是 Python 中的一种比较流行的图形编程接口。Tkinter 模块（“TK 接口”）是 Python 的标准 Tk GUI 工具包的接口。TK 和 Tkinter 是为数不多的跨平台的脚本图形界面接口，被应用在多个系统中，如 Unix、Windows 和 Macintosh 系统。

Tkinter 包含了若干模块。 Tk 接口被封装在一个名为 _tkinter 二进制模块里（ tkinter 的早期版本）。这个模块包含了 Tk 的低级接口，因而它不会被程序员直接应用。它通常表现为一个共享库（或 DLL 文件），通过 Tkinter.py 来使用，但在一些版本中它与 Python 解释器结合在一起。

不管你是主动还是被动获得，Tkinter 中的 GUI 总是有一个 root 窗口，主窗口就是程序开始运行的时候创建的，在主窗口中你通常放置主要的部件。另外，Tkinter 脚本可以依据需要创建很多独立的窗口，主要的方法就是通过创建 Toplevel 对象。每一个 Toplevel 对象都创建一个显示的窗口，无须通过 mainloop 方法调用。

在 Tkinter 编程中应注意的问题

在 Tk 接口的附加模块中，Tkinter 包含了一些 Python 模块，保存在标准库的一个子目录里，称为 tkinter。 其中有两个重要的模块，一个是 Tkinter 本身，另一个叫做 Tkconstants。前者自动导入后者，所以你如果使用 Tkinter，仅仅导入一个模块就可以。

```
    import Tkinter
```

或者

```
from Tkinter import *
```

那么让我们来体验一下 Tkinter 的图形设计的魅力吧。

下来我们就来编写一个最常见的程序“Hello,world”。

```
# -*- coding: utf-8 -*-
from Tkinter import *
#导入 Tkinter 模块

root = Tk()
#创建一个 root
w = Label(root,text = "Hello,world!")
#w 是 root 的子窗口,而 text 是 w 的一个选项,表示 w 中要显示的内容
w.pack()
#在 pack 后,计算好 Label 的大小,最终显示在屏幕上
root.mainloop()
#mainloop()除里内部的 widget 的更新,和来自 Windows Manager 的通信
```

运行程序：

```
~$ python pcs-217-1.py
```

运行结果如图 PCS217-1 所示：

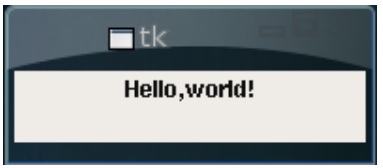


图 PCS 217-1　hello world 示例 1

当然，对于上面这样的小程序，我们可以直接将其写在一个文件里。但是对于一个较大的程序，最好将它写在一个类中，如下：

```
# -*- coding: utf-8 -*-
from Tkinter import *

class App:
  def __init__(self, master):
     #Frame widget
     frame = Frame(master)
     frame.pack()

     self.button = Button ( frame, # master widget
                           text="QUIT",
```

```
                          fg="red",
                          command=frame.quit
                          )
        self.button.pack(side=LEFT)
        self.hi_there = Button(frame,
                          text="Hello",
                          command=self.say_hi
                          )
        self.hi_there.pack(side=LEFT)
    def say_hi(self):
        print "Welcome to the world of Python!"
root = Tk()
app = App(root)
root.mainloop()
```

运行程序：

```
~$ python pcs-217-2.py
```

如果你按“hello”的按钮，会在控制台上打印“Welcome to the world of Python!”

运行结果如图 PCS217-2 所示：

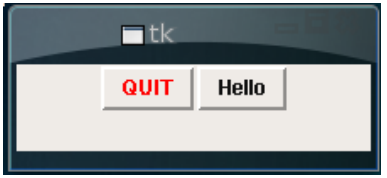


图 PCS 217-2　hello world 示例 2

Geometry Manager（几何管理器）

在图形设计过程，经常会涉及一些布局的美化和合理放置，从而使效果更加友好。为了解决这个问题，在 Tkinter 中引入了这样一个概念。

Tk 提供了几个 Geometry Manager，主要是为了帮助开发者完成在布局中的 widget 的放置。一般 Tk 有三种几何管理器：Pack、Grid、Place，这里只着重介绍一下 Pack 管理器。

Pack 管理器的使用方法很简单，在 w.pack(options)中，w 是一个 widget。常用的 options 如下所示。

- side 表示把 w 放在那个边上，可以是 TOP，BOTTOM，LEFT，RIGHT。
- padx 和 pady 表示 parcel 的每一个边和 w 之间预留的空间。

- ipadx 和 ipady 表示 w 的每一个边和 w 内包含的内容之间的预留的空间，w 会因此变大。
- fill 可以是 None、x、y、both。如果 parcel 太大了，那就会根据 fill 的值来增加子窗口的大小：

 None 表示维持子窗口原来的大小；
 X 表示只扩大子窗口的宽度；
 Y 表示只扩大子窗口的高度；
 BOTH 表示同时扩大子窗口的宽度和高度。

值得注意的是，fill 是指子窗口的大小和 parcel 大小的关系。决定子窗口怎样改变自己的大小，来适应 parcel 的大小。

思考

1. 在利用 Tkinter 来设计图形界面时，如果不使用 root()，会出现什么结果？
2. 和 wxPython 等 Python 图形模块相比，Tkinter 的使用有哪些优缺点？
3. 图形界面的异常处理于一般异常有何不同？必须考虑哪些具体问题？

框架篇

PCS300	CherryPy	324
PCS301	Karrigell	327
PCS302	Leo	334
PCS303	MoinMoin	347
PCS304	Python Web 应用框架纵论	352

PCS300　CherryPy

啥是 CherryPy

CherryPy 是个非常简洁易用、面向对象的 Web 框架。咦，面向对象不是用来形容程序设计语言的，怎么用来形容 Web 框架了？看下面你就会明白了。

使用 CherryPy 编写 Web 程序的过程，就是编写一组对象，再通过 CherryPy 发布出去的过程。这样一组对象通过属性的引用构成了一个图，我们只须把一个根节点交给 CherryPy。这样 CherryPy 就会将用户请求的 URL 映射到这些对象，解析 URL 的时候就等于是在这个对象图中漫游。最终 CherryPy 确定目标对象和相应方法，将 URL 的参数和 POST 参数根据名字映射到方法的参数，然后调用其方法，并返回其结果给客户端。

使用 CherryPy 编写 Web 程序的体验是非常舒服自然的，下面就跟我来尝试一把吧。

下载安装

从 *http://www.cherrypy.org/wiki/CherryPyDownload*（精巧地址：*http://bit.ly/41FZYA*）找到最新版本的下载地址并下载，解压，然后和几乎所有 python 模块的安装一样，执行 `python setup.py install`，则大功告成，可以开始写程序了。

开始写程序

我们就从 hello world！开始。编写 hello.py ，内容如下，保存成 utf-8 格式：

```
# -*- coding: utf-8 -*-
import cherrypy

class HelloWorld:
    # 定义一个 index 的方法
    def index(self):
        return "Hello world!"
    # 将 index 方法发布到 web
    index.exposed = True

# 把根对象交给 cherrypy（不过这里总共就一个对象而已）
cherrypy.quickstart(HelloWorld())
```

然后打开命令行，直接执行 `python hello.py`。现在打开浏览器访问 *http://localhost:8080/index*，就可以看到上面 `index` 方法返回的那句话了，也可以直接访问 *http://localhost:8080/* ，index 是默认调用的方法。

再来看个稍微复杂点的吧：

```
# -*- coding: utf-8 -*-
import cherrypy

class OnePage(object):
    def index(self):
        return "one page!"
    index.exposed = True

class HelloWorld(object):
    # 通过属性引用另一个对象，cherrypy 解析 URL 的时候就会沿着属性引用的道路前进。
    onepage = OnePage()

    def index(self):
        return "hello world"
    index.exposed = True

    # 定义了一个参数，cherrypy 会将 URL 中的同名参数映射过来
    def hello(self, name):
        return "hello %s" % name
    hello.exposed = True

cherrypy.quickstart(HelloWorld())
```

执行后，访问 *http://localhost:8080/*和 *http://localhost:8080/onepage* 可以分别看到 hello world 和 one page!。访问 *http://localhost:8080/hello?name=you* 也可以看到返回了 hello you。

上面处理的是 GET 参数，也就是 URL 中带的参数，CherryPy 对 POST 参数和 GET 参数一视同仁，一律映射为方法的参数，POST 的处理其实是完全一样的。

另外在实际的 Web 应用中，当然不能把页面的 html 全部塞到 Python 代码里面来，实际上有很多专门的模板可以用，详见 PCS 304“Python Web 应用框架纵论”。

关于 CherryPy 的详细内容，读者可以参考 CherryPy 的官方文档。

- 访问地址：*http://www.cherrypy.org/wiki/TableOfContents*

 精巧地址：*http://bit.ly/425soh*

PCS301　Karrigell

安心小推车

概述

Karrigell 是一个易用的 Python Web 应用框架，使用方式清晰直观。它独立于任何数据库、ORM 和模板引擎，也可以让开发人员选择不同的编码风格。并且内置有一个功能强大的 Web 服务器，还内置了一个纯 Python 的对象型数据库 PyDbLite（之前是 Gadfly，再之前是 buzhug），其访问地址为 *http://quentel.pierre.free.fr/PyDbLite/index.html*（精巧地址：*http://bit.ly/4ebSc8*）。而且，可以通过配置来与其他 Web 服务器（如 Apache、Xitami、LightTPD、Ngnix 等）一起工作，还可以结合 Python 的其他数据库（sqlite、mySql、PostGreSQL、ZODB 等）接口来创建数据库应用程序。它还——

- 有内置的 session 处理。
- 有认证机制。
- 是多语言支持的（从 2.1 开始）。
- 自带模板处理功能（Cheetah）。

应用

故事 CDays/KDays 都是选择 Karrigell 进行 Web 应用组织的，这里精要地指出基于 Karrigell 进行 Web 应用开发的基本形式。

下面是“Hello world”程序使用的 5 种不同的编码方式。

- Python 脚本（文件:hello.py）

```
1 print "Hello, world ! "
```

- Karrigell 服务脚本 （文件:hello.ks）

```
1 def index():
2     print "Hello, world ! "
```

- Python 嵌 HTML （文件 hello.hip）

```
"Hello, world ! "
```

- HTML 嵌 Python （文件:hello.pih）

```
Hello, world !
```

- CGI 脚本 （文件:hello.py 在目录:cgi-bin）

```
1 print "Content-type: text/html"
2 print
3 print "Hello, world !"
```

小实例

如果要在页面中打印出 0 到 9 的数字序列，在 Python 中 5 种不同的编码方式分别为：

- Python 脚本（文件:squares.py）

```
1 print "<h1>Squares</h1>"
2 for i in range(10):
3     print "%s :<b>%s</b>" %(i,i*i)
```

- Karrigell 服务脚本 （文件:squares.ks）

```
1 def index():
2     print "<h1>Squares</h1>"
3     for i in range(10):
4         print "%s :<b>%s</b>" %(i,i*i)
```

- Python 嵌 HTML （squares.hip）

```
"<h1>Squares</h1>"
for i in range(10):
    "%s :<b>%s</b>" %(i,i*i)
```

- HTML 嵌 Python （文件:squares.pih）

```
<h1>Squares</h1>
<%
for i in range(10):
```

```
    print "%s :<b>%s</b>" %(i,i*i)
%>
```

- CGI 脚本 （文件:squares.py 在目录:cgi-bin）

```
1 print 'Content-type: text/html'
2 print
3 print "<h1>Squares</h1>"
4 for i in range(10):
5     print "%s :<b>%s</b>" %(i,i*i)
```

综述

事实上可以任意选择写 Python 脚本的方式。比如，可以使用普通的 Python 脚本，也可以使用 Karrigell，Karrigell 服务脚本是一种 Python 脚本，它的每一个函式都匹配一个不同的 URL：`foo.ks/bar`，匹配 foo.ks 脚本中的 bar()函式（如果没有指定函式，默认使用 index()函式）。

另一种写 Python 脚本的方式是 Python 嵌 HTML（HTML inside Python），当它运行时遇到 print 语句就会把要输出的字符串发送给客户端浏览器。

HTML 嵌 Python（Python inside HTML）是一种非常像 ASP、JSP、PHP 的写法，Python 写在 HTML 页面的< % %>标记中。

还可以直接使用 CGI 脚本，参见 Python 标准文档的 `cgi module` 章节。

表单处置

Python 代码在一个包含 HTTP 环境、表单字段、有自定义异常的命名空间下运行。当一个表单包括字段`<INPUT name="myfield">`时，它的值可以在脚本中使用`_myfield` 来得到。

身份认证和 Session

可以在脚本中使用两个叫做 Authentication 和 Session 的函式来处理。 Authentication 的第一个参数是一个测试函式，用来检查是否接受输入的 `AUTH_USER` 和 `AUTH_PASSWORD`。Session()用来初始化一个 session 对象并设置或读取它的属性值或者得到一个 session。

文件包含内容

`Include(file_or_script)`函式在当前脚本插入脚本或者文件的输出结果，例如它可用在页头或者页尾。

实例

HYRY Studio 是一位在日本的朋友独立创建的综合个人网站（*http://hyry.dip.jp/index.py*，精巧地址：*http://bit.ly/2hfbkQ*），可以在首页看到网站的发展历程（このサイトの技術と開発履歴），它全部使用 Karrigell 架构！其中还独立开发了 HYRY Blog（*http://hyry.dip.jp/blogt.py?catlog=Python 学习*，精巧地址：*http://bit.ly/2qgnkh*）。

问题

Karrigell 自带的发布服务运行简单，零配置，在网站开发或是运营初期完全不用通过 CGI/fastCGI/SCGI/WSGI 等接口和专业 Web 服务器绑定，使用它自个儿就成。但是，问题在于，总是要开个命令行来启动，关闭命令行窗口，服务也就关闭了，那么如何方便地将 Karrigell 应用作为系统服务进行发布呢？

在 MS Windows 下面怎样弄？甭琢磨了，不会轻易让你这么玩的。那在 Unix 下面呢？太自然了，和一切服务一样！伪造成服务类的脚本来启动就可以了！

下面以 FreeBSD 为例，说明配置步骤。

1. 创建启动脚本（比如叫 runK.sh）：

```
#! /bin/sh
## 先进入对应目录
cd /path/to/Karrigell
## 使用 daemon 命令来包裹并记录 Karrigell 的运行 pid
/usr/sbin/daemon -p /var/run/karrigell.pid /usr/local/bin/python
Karrigell.py 2>/var/log/K.log &

exit  0
```

2. 在`/usr/local/etc/rc.d`中创建 rcNG 脚本（比如说叫 karrigell.sh）

```
#! /bin/sh
#
# PROVIDE: karrigell
```

```
# REQUIRE: DAEMON
# KEYWORD: shutdown
#
# DO NOT CHANGE THESE DEFAULT VALUES HERE
# SET THEM IN THE /etc/rc.conf FILE
#
karrigell_enable=${karrigell_enable-"YES"}
karrigell_flags=${karrigell_flags-""}
karrigell_pidfile="/var/run/karrigell.pid"
## 指向的 pid 和启动脚本中声明的要一致

. /etc/rc.subr

name="karrigell"
rcvar=`set_rcvar`
command="/path/to/u/runK.sh"
## 使用绝对路径声明启运脚本
procname="/usr/local/bin/python"
## 声明运行的环境，用绝对路径说明 Python 的运行环境

load_rc_config $name
pidfile="${karrigell_pidfile}"
start_cmd="echo \"Starting ${name}.\"; /usr/bin/nice -5 ${command}
${karrigell_flags} ${command_args}"

run_rc_command "$1"
```

3. 最后在/etc/rc.conf 中追加

```
## 打开 rcNG Karrigell 识别
karrigell_enable = "YES"
```

齐了！现在可以像普通服务一样进行标准的服务开启和关闭了：

```
# /usr/local/etc/rc.d/karrigell start
# /usr/local/etc/rc.d/karrigell stop
```

探讨

Karrigell 可以说是为了最简单和自然地进行轻型单一功能型网站开发而创建的框架，它所关注的是简单的开发和调试，而不是高性能的数据库操作和网络响应。那么，我们是否真的不能用 Karrigell 来架设高负荷的网站了？

高性能化 Karrigell

1. 用 mod_wsgi 来提升性能可参见“How to use Karrigell with mod_wsgi（如何让 Karrigell 和 mod_wsgi 一同工作）”，这篇文章对 Apache 的 WSGI 模块如何令 Karrigell 应用成倍地提高响应速度进行了探索。

 访问地址：*http://code.google.com/p/modwsgi/wiki/IntegrationWithKarrigell*

 精巧地址：*http://bit.ly/40AJd*

2. 如果反感 Apache 的臃肿，那么可以使用 Karrigell + PyProcessing,Very good and Strong！

 访问地址：

 http://wiki.woodpecker.org.cn/moin/KarrigellWithPyProcessingVeryGoodStrong

 精巧地址：*http://bit.ly/1HvKDU*

 可以直接定制 Karrigell 服务脚本，引入多线程机制从而获得 50 倍的提速！

 注意：这里指的是在 FreeBSD 环境中，在 MS Windows 环境中因为没有真正的多线程支持环境，是无法轻易做到这点的。

3. 读者也可以尝试使用 Stackless Python 来启动 Karrigell，在不改动任何一行代码的情况下也可能获得速度的提升。

4. 进一步，也可以尝试使用 Psyco/PyRex 等脚本增强支持来提高速度。

小结

由于 Karrigell 出现得比较早，网站中散见有各种程度的实际使用体验，建议读者进一步参考。

Karrigell 与其他平台的不同在于：

- “Karrigell 学习”我为什么选择了 Karrigell - limodou 的学习记录

 访问地址：*http://www.donews.net/limodou/archive/2005/04/23/347349.aspx*

 精巧地址：*http://bit.ly/1ZodzQ*

- Python 下的轻型 web framework - limodou 的学习记录

 访问地址：*http://www.donews.net/limodou/archive/2005/03/14/302380.aspx*

精巧地址：*http://bit.ly/37X6NY*

还有更多有关学习 Karrigell 的记录，比如来自 ChumpKlutz（朽木）原创翻译：

- Karrigell 官方文档中文版

 访问地址：*http://blog.csdn.net/ChumpKlutz/category/243254.aspx*

 精巧地址：*http://bit.ly/4DWPHf*

- Karrigell Web 开发入门系列

 访问地址：*http://blog.csdn.net/ChumpKlutz/archive/2007/09/12/1781990.aspx*

 精巧地址：*http://bit.ly/3ADgnA*

这两份翻译的 PDF 版可到以下网站下载：

访问地址：*http://code.google.com/p/gopython/downloads/list*

精巧地址：*http://bit.ly/tODOJ*

PCS302　Leo

文学化编程环境

Leo 是——

Leo 可以是一个通用的数据管理环境；

Leo 可以是一个柔韧的程序/文本，或是其他数据的综合浏览器；

Leo 可以是一个项目管理器；

Leo 可以是一个提纲式的程序编辑器；

……

这个纯 Python 的编辑环境用最简单的外观实现了如此丰富的使用方式和编程思想！以至于无法简单地定义 Leo 到底是什么！

不过可以肯定一点：Leo 不是 IDE！

IDE 之殇：

IDE（Integrated Development Environment）集成开发环境，维基百科对其定义：*http://zh.wikipedia.org/wiki/集成开发环境*（精巧地址：*http://bit.ly/xouAm*）。

IDE 指的是在同一界面综合了编辑、调试、运行、外部环境交互等开发支持的环境，Eclipse 就是个经典的 IDE 环境。不过从笔者的角度，认为 IDE 是有害的，它令编程人员远离了真正的细节，以为编程就是控件的组合，放弃了对最终产品的多方面调节，而且，IDE 中的自动提醒/函式完成等辅助工具，实际上起到的只是中断思路，岔开注意力，影响思考的作用！所以，我们推荐使用 Leo 等单纯的文本编程环境来编程，通过函式复用或是合理模块化来加速开发，而不是盲目依赖 IDE 辅助开发支持！

另外，行者有妙文流传——“IDE 综合症和摄影”。

访问地址：*http://wiki.woodpecker.org.cn/moin/MiscItems/2008-09-28*

精巧地址：*http://bit.ly/taeut*

概述

Leo 是基于 Noweb 数据工具的文学化编程环境。Noweb 是一个已维护了 15 年的成熟自由软件，用以支持语言无关的文学化编程。

官方网站：*http://www.cs.tufts.edu/~nr/noweb/*

精巧地址：*http://bit.ly/3OvVCf*

由于 Leo 通过友好的界面包裹了 Noweb 的功能，一般不必对 Noweb 有太多的认识。

而文学化编程（Literate Programming）是 TEX 的发明人 Donald E. Knuth 在 1984 年提出的概念：

源代码只是构成程序的一小部分；程序的真正主体是对它的算法、结构、目的和用法的描述——实现它的源代码并不是主要的。所以，文学化编程关注人是怎么理解和构建程序的，而不是像其他主流开发环境那样，期望人来理解程序应该怎么样运行和组织！

定义网址：*http://en.wikipedia.org/wiki/Literate_programming*

精巧地址：*http://bit.ly/aCxZG*

官方网站：*http://www.literateprogramming.com*

精巧地址：*http://bit.ly/3I9Rxg*

所以，Leo 的名字含义是 Literate Editor with Outlines（基于提纲的文学化编辑器），但是，Leo 最常见的使用方式就是用来文学化地组织软件开发工程。

应用

通过实际操作，可以快速体验到文学化编程的魅力。

安装

首先得确认 Python 环境安装好了，测验标准就是在命令行状态中输入 `python` 系统就返

回一个标准的 Python 交互环境。去官方网站 *http://sourceforge.net/projects/leo*（精巧地址：*http://bit.ly/2mefKP*）下载，当前最新版本是 Leo-4-5-1-final.zip（2008-09-14），解开压缩后，复制到任何你希望的目录，不用安装，在命令行环境直接运行。

```
#GNU/Linux;Unix 中
$ python /path/to/leo/launchLeo.py
#Windows 中:
C:\> python C:\path\to\leo\launchLeo.py
```

这样，就可以获得 Leo 环境了！当然，可以创建快捷方式（不通过命令行）来快速调用 Leo。

Tkinter 是 Python 的 GUI 开发内置组件，一般 GNU/Linux 软件仓库都有，但是可能没有被默认安装。安装 Tkinter：

```
$ sudo apt-get install python-tk
```

默认的 Leo 界面是由**提纲窗口**、**正文窗口**和**提示窗口**组成的。基本的操作就是在提纲窗口中组织章节/节点结构，在正文窗口中组织对应章节/节点的内容。

基础操作

在 Leo 中核心的和唯一的操作单元就是提纲节点（outline）！自然地，最基础和关键的操作就是对提纲节点（outline）进行的操作。

1. 创建一个节点，快捷键是 `Ctrl+I`（如图 PCS302-1 所示）。点击提纲节点或是 `Ctrl+H` 就可以对节点名称进行修订。

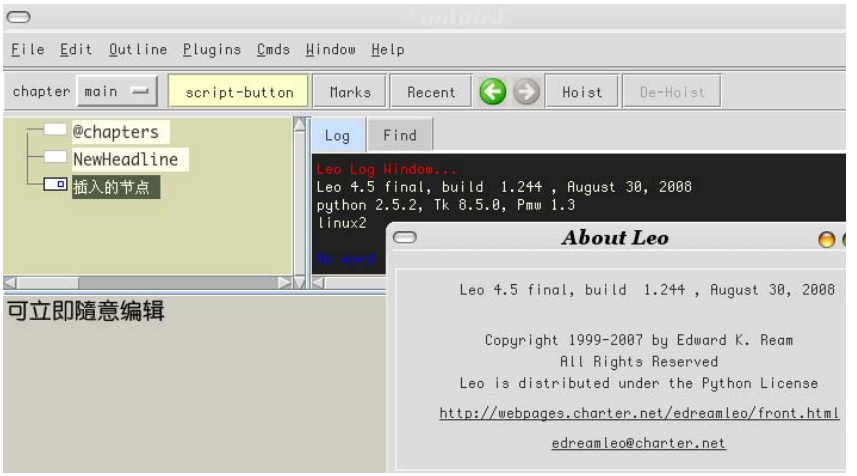


图 PCS302-1　插入节点

2. 调整一个节点的层级，快捷键是 `Ctrl+U`、`Ctrl+D`、`Ctrl+L`、`Ctrl+R`，对应地将节点向上下左右移动。
3. 复制/粘贴一个节点，快捷键是 `Ctrl+Shift+C`、`Ctrl+Shift+V`、`Ctrl+Shift+X`，对应的就是对节点进行复制/粘贴/剪切。
4. 创建一个节点的克隆（clone），快捷键是 `Ctrl+`（如图 PCS302-2 所示）。

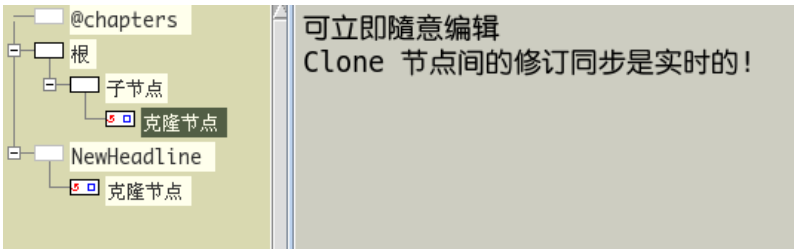


图 PCS302-2　提纲树中含有克隆节点的情景

克隆的意思就是不论克隆节点有多少，在哪儿，其实都是同一节点的自动复制，任何一个节点的修订，将立即导致所有克隆节点的同步变化，这对编辑中，部分代码需要在不同文件或是函式中保持一致是多么的人性和可亲哪！

5. 移动当前的编辑焦点到不同节点，快捷键是 Alt+↑/↓/←/→，对应移动焦点到 上/下，左/右（上一级/次一级）节点。其中 Alt+←/→也同时自动对当前层级的节点进行折叠和展开的操作。

综述

提纲节点（outline）在文学中就是自然的章节和段落，在编程中就是类似/函式/逻辑段，在 Leo 中，节点是唯一关注的单元，而不论节点中的内容是什么！这充分表达了文学化编程中，关注结构不关注代码的思想！

对于 Python 行者来说，Leo 中的节点操作有以下几个让人感觉舒服的地方：

1. 节点名可以使用中文；
2. 子节点的数量不限！（界限是自个儿机器的内存）；
3. 节点可以包含任何文本，不论这些文本来自哪个文件（这方面在下文详述）；
4. 节点可以通过各种算子原地引用在各种缩进的文本处，且可以在输出时继承挂接点的缩进，然而在节点正文中不用对应地缩进！

文件

Leo 只是个文学化编辑环境，所有组织的内容，总得输出成标准的文件，来给其他应用使用/运行，所以，文件级别的操作也是非常必要和基础的。

导入

在 `File->Import` 菜单有很多导入方式，但是常用的就是两种：`Import to @file`（导入@file 容器）和 `Import to @file`（导入@root 容器）。都会自动根据导入的文件类型自动解析并将相关内容装配到对应的有良好命名的节点中！

导出

在 `File->Export` 菜单有很多导出方式，不过，一般都是使用相应策略的文件导入“算子”，令相关的节点自动组织成不同的文件输出，而且，导出动作的快捷键，也可以和保存 Leo 文件一样复用 `Ctrl+S` 就好！

算子

Leo 当然不是专门用来写小说的，所以，有一些特殊的约定字串——“算子”包含在正文或是节点中，协助我们高效地使用文学化思想操作大量的文本，笔者在此介绍几个最最常用的算子，进一步的可以参考 Leo 自带的文档（如图 PCS302-3 所示）。

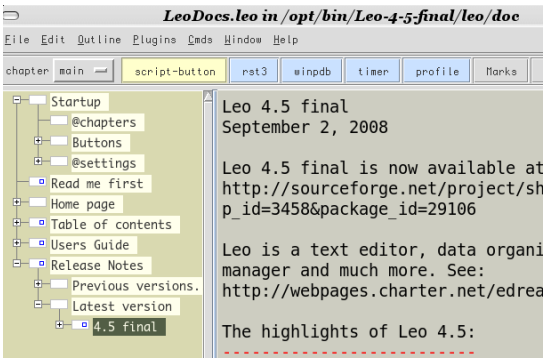


图 PCS302-3　Leo 安装包中内置的帮助文档树

@nosent：文件净导出算子（输出请见图 PCS302-4）

位置：节点中

格式：@nosent+空格+文件名（可包含全路径）

作用：将所有相关节点整理后，输出没有 noweb 节点说明注释格式的干净文本文件。

局限：输出的文件不含 Leo 中的定制结构信息，Leo 文件丢失，将无法自动复原文学化章节关系。

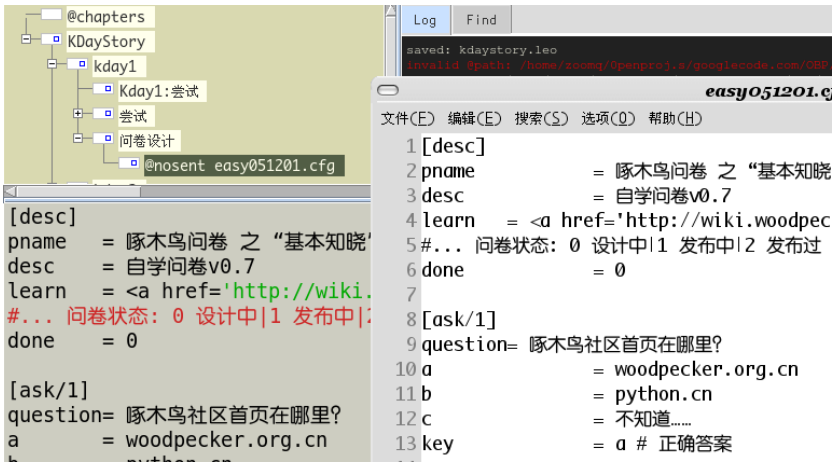


图 PCS302-4　Leo 使用@nosent 算子的输出

@file：文件注释导出算子（输出请见图 PCS302-5）

位置：节点中

格式：@file+空格+文件名（可包含全路径）

作用：将所有相关节点整理后，输出含有 noweb 节点说明注释格式的文本文件。

局限：输出的文件含有 Leo 中的定制结构信息，可以在其他 Leo 文件中导入。但是其他外成员不使用 Leo 时，过多的 noweb 注释是种干扰。

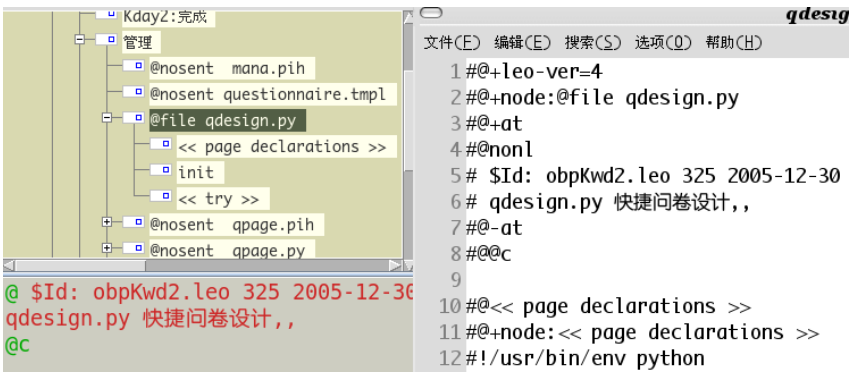


图 PCS302-5　Leo 使用@file 算子的输出

@path：目录声明算子（输出请见图 PCS302-6）

位置：节点正文

格式：@path+空格+相对或是绝对路径

作用：配合各种文件导出算子指示 Leo 将对应节点输出到何处。

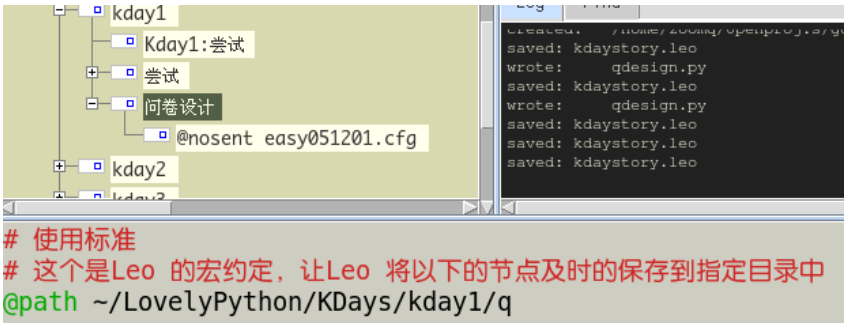


图 PCS302-6　@path 配合@nosent 算子进行文件输出

@others：递归包含算子

位置：节点正文

格式：@others 单独一行

作用：指示 Leo 将当前节点之下所有子节点包含到当前位置

@ ...@c：多行注释算子

位置：节点正文

作用：指示 Leo 将@和@c 中包含的所有内容，作为注释输出

@ 表示注释开始；@c 表示之下是代码

<<引用>> ：引用算子

位置：节点正文

作用：指示 Leo 将节点名是 <<引用>> 的节点内容引用到当前位置，可以多次使用。引用的节点名不限长度。

@language python：语言算子

位置：节点正文，一般在首部

格式：@nosent+空格+文件名(可包含全路径)

作用：指示 Leo 使用什么语言来解析之下的代码。

@nocolor/@color：语法色算子

位置：节点正文，一般在首部

格式：@nocolor 或是@color 单独一行

作用：指示 Leo 对之后的文本是否尝试进行语法颜色的处理

Leo 可以智能地根据文件后缀，或是语言算子，自动解决十数种程序语言的语法颜色！

编程

Leo 作为编程环境非常有个性，其感受和其他任何编程环境都不同！需要适应，这里笔者根据切身体验指出几个必须跨过的习惯门槛，分享使用 Leo 进行 Python 编程的快乐体验。

Leo 的不便之处

并不是说 Leo 编程环境全部都是好用的，这里就先指出使用 Leo 时比较麻烦的地方。

1. Leo 不是 IDE，没有单步追踪、自动化完成提醒、包函式提醒等各种 IDE 的高级自动化功能！
2. Leo 不是任何人都喜爱的，所以，在团队协同中，如果不是所有人都使用 Leo，则在配合时，如果统一使用包含 Leo 结构化信息的 noweb 注释文本的输出，对于不使用 Leo 的伙伴就是种困扰，同理，如果统一使用没有 noweb 注释的干净文本，那么使用 Leo 的伙伴将无法方便地在 Leo 环境中及时看到其他人的修订。

Leo 的便捷之处

除以上提及的主要不便外，使用 Leo 可以帮助我们真正高效地理解和控制代码的结构，让人体会到自如的重构快感！

1. Leo 作为工程组织环境，可以自然地将多个工程、多个目录、多个文件、综合组织在一个界面中！

 因为，Leo 的基本单位是提纲节点（outline），而节点包含什么或是输出成什么，对于 Leo 没有任何关系！所以，在 Leo 中，可以方便地做到多个工程中的文件/类/函

式/代码段复用！

2. Leo 作为文学化编辑环境，鼓励从伪代码开始组织应用！即，完全可以先通过节点的设计和组织，建筑起完整的功能模块框架，通过函式名等的预先设计，从文字方面确认功能如何完成后，再逐一使用真实代码完成。进一步，这也鼓励先在设计好的空函式部分，对应设计好测试用例代码，进行 TDD 测试驱动式开发！

3. 使用 Leo 可以帮助我们快速地使用“丑陋”的代码，先将功能“堆”出来，然后快速重构成高效合理的对象化代码。一开始不用顾虑什么设计模式，就用直觉式的代码，一段段将功能完成。这其中必然会产生高过 50 行以上的巨型函式，但是在 Leo 的帮助下，可以将函式根据我们自个儿的设计/理解，划分成任意层次的逻辑结构，这样一来：

 （1）可以安全地将需要实现的代码封闭在一个个小节点中进行调试，不会像使用 IDE 中那样，经常不小心错误移动到相似的代码段中编辑；

 （2）编辑结构可以使用中文来命名，无形中将功能的实现思路也记录了下来，方便在有空时，根据原有设计，逐一函式化，成为精简可控的小函式。

问题

学习使用 Leo 必然要面对很多问题，这里分享对其中两点问题的应对方法。

心理问题

“使用 Tk 开发的哪，界面这么简陋！”

“什么是文学化编辑？我只是需要个有自动完成提示的环境！”

“为什么要调整和分享节点呐？！”

这些问题是很多尝试 Leo 后立即放弃的朋友们的反应，其实这就是使用不同于 VC 等等 IDE 环境进行编程的必然门槛。

心理习惯

人都有先入为主的习性，对于不熟悉的环境，总是期望使用既有的经验进行对比/拟合，一旦没有可比性就会失措、失望。然而，Leo 的文学化编辑思想和操作是本质的飞跃，

是和我们现有的体验完全不同的。所以，建议先不要试图理解，而是先习惯基本操作，在小的项目中先尝试运用 Leo 进行组织，在使用中体验“文学化编辑”，慢慢地，就会有一天猛然一贯而通，使用起节点来如同呼吸般自然和高效了！

中文问题

Leo 是使用 Python 基于 Tk 框架开发的，而 Tk 是一个古老而简洁的桌面图形化软件支持库，以往的版本对中文的支持，在非 M$平台一直有问题，这里快速说明一下 Ubuntu（一种 GNU/Linux 操作系统）中 Leo 中文显示问题的解决方法。

现象

无法使用输入法输入中文；字体可选择种类非常少；边缘显示有毛刺现象，这些都可以利用 Python 来探测。

```
  $ python
Python 2.5.2 (r252:60911, Jun 21 2008, 09:47:06)
[GCC 4.2.3 (Ubuntu 4.2.3-2ubuntu7)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import Tkinter
>>> print Tkinter.TclVersion
8.4
```

如果系统输出 Tk 版本是 8.4 的话，证明 Tk 框架是不支持中文的老版本。

思路

感谢自由软件社区，这个问题早在两年前就由台湾的 aMSN 社区（*http://www.amsn-project.net*）在开发过程中解决了，而且补丁被包含在 Tk8.5 版本中。但是在 Ubuntu 等操作系统中，Python 预编译版本都是绑定的 Tk8.4 版，所以，只要重编译 Python 将 Tk 最新版本包含进来就可以完善中文的支持。

处置

安装 Tk8.5，在 Ubuntu 的软件仓库中就可找到，所以，可以使用“新立得”工具快速安

装，命令行所示。

```
$ sudo apt-get install tk8.5 tk8.5-dev
```

从 Python 官方网站下载对应本地 CPU 的源代码，下载地址：*http://www.python.org/ftp/python/2.5.2/*（精巧地址：*http://bit.ly/2cLwy1*）

依次进行以下操作。

```
$ tar xjv Python-2.5.2.tar.bz2
$ cd Python-2.5.2
$ ./configure --prefix=/usr --enable-unicode=ucs4
$ make
$ sudo make install
```

完成编译和安装，然后替换原先的 Python 调用。

```
$ cd /usr/bin
$ sudo rm python
$ sudo ln -s python2.5 python
```

再进行之前的测试，就应该可以看到 Tkinter.TclVersion 的值变成了 8.5，而且可以从菜单 Cmds->Picklers->Show-Fonts 中看到之前无法选择到的各种中文字体了！

提醒：如果处理后，还有问题，请及时升级 Leo 到最新版本，到 2008 年 09 月 02 日时，Leo 已经是 4.5.0 版了，新版本对各种特性都有精心提升，可以支持最粗暴的操作，建议及时安装体验。

瞒天过海

由于 Tk8.5 对 Tk8.4 的 API 改动并不大，所以其实可以瞒天过海用 Tk8.5 直接代替 Tk8.4 而不需要重新编译任何东西（这是行者 Jiahua Huang <jhuangjiahua@gmail.com>探索出的好招）！当然是指 GNU/Linux 类的自由软件平台中，毕竟这是允许用户全权操控的环境哪。可以在终端窗口粘贴执行以下命令。

```
# 安装 Tk8.5
sudo apt-get install tk8.5 tcl8.5

# 先备份
sudo cp /usr/lib/libtcl8.4.so.0 /usr/lib/libtcl8.4.so.0.old
sudo cp /usr/lib/libtk8.4.so.0  /usr/lib/libtk8.4.so.0.old
sudo cp /usr/lib/python2.5/lib-dynload/_tkinter.so
/usr/lib/python2.5/lib-dynload/_tkinter.so.old
```

```
# 再用 Tk8.5 覆盖 Tk8.4
sudo cp /usr/bin/tclsh8.5 /usr/bin/tclsh8.4
sudo cp /usr/bin/wish8.5  /usr/bin/wish8.4
sudo cp /usr/lib/libtcl8.5.so.0 /usr/lib/libtcl8.4.so.0
sudo cp /usr/lib/libtk8.5.so.0  /usr/lib/libtk8.4.so.0

# 篡改 "Tk8.4" 的版本号
sudo sed -i 's/8\.5/8.4/g' /usr/lib/libtk8.4.so.0
sudo sed -i 's/8\.5/8.4/g' /usr/lib/libtcl8.4.so.0
```

得了您呐！齐活儿！

探讨

Leo 社区是稳健活跃的，Leo 的开发和升级也是积极的，Leo 的 FANs 们已经发展出了不少高级或是神奇的使用方法:

LeoN

协同网络化的 Leo！LeoN 的使用情景请见图 PCS302-7。

官方网站：*http://ryalias.freezope.org/souvenirs/leon*

精巧地址：*http://bit.ly/EIR8I*

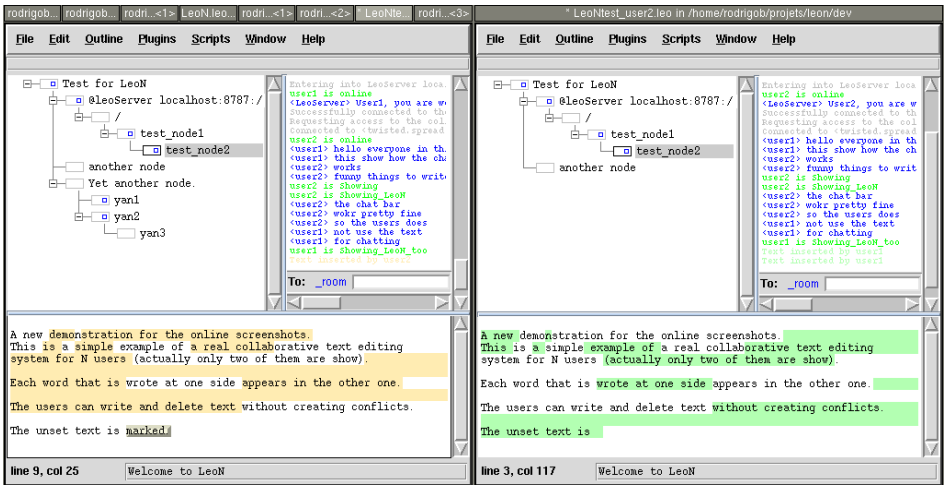


图 PCS302-7　LeoN 的使用情景

通过 LeoN 可以支持多个开发人员，在网络中同时协同同一个 Leo 节点！

插件

Leo 从一开始就提供了标准的接口，在手册中有专门一章进行描述“ Chapter 7: Scripting Leo with Python”其访问地址：*http://webpages.charter.net/edreamleo/scripting.html*（精巧地址：*http://bit.ly/4lYHIW*）

在社区维基中收集有最成熟和实用的插件，其访问地址：*http://leo.zwiki.org/Plugins*（精巧地址：*http://bit.ly/390i9b*）

小结

使用 Leo 是绝对不同以往的编程体验，但其本身就是使用纯 Python 实现的，值得学习和体验！

PCS303 MoinMoin

纯 Python 流行 Wiki 系统！

概述

Wiki

Wiki 就是维基，名字来源自夏威夷语的“wee kee wee kee”，意思就是“快”，本质上是种快速内容发布系统，但是有别于其他的系统，它有其非常个性的约定：

所有人可以修订所有页面！这是维基最最本质的特征，也因为这个，引发了维基的爆发式发展，成为流行的知识管理平台、共笔系统。

通过 WikiName 的约定，可以快速相互链接！WikiName 就是由 2 个以上首字母大写的单词组成的字串，会由维基系统自动解析为一个指向维基页面的链接，如果该页面不存在就会立即建立一个空页面，等待读者自行完善。

使用 WYTIWYG——所想即所得思想指导的结构化文本来组织内容，而不是 HTML 之类的“所见即所得”式的富文本管理器。

MoinMoin

MoinMoin 是一个基于 Python 环境的 wiki 引擎程序，支持包括中文在内的多语种特性，是遵循 GNU GPL 的开源项目，启动于 2000 年 7 月 20 日，最初由 JürgenHermann 撰写。

可运行在 Windows、GNU/Linux、BSD、UNIX、OS X 等环境下。目前能够处理英文、德文、繁简体中文、日文、俄文等约 21 种语言。

MoinMoin 的特点:

1. 完全使用文件来存储内容，不使用数据库；
2. 实现了全部 Wiki 规范，Unicode 编码支持多语种；
3. 完整实用的 wiki 文本约定，编辑规则比较轻巧易学，包含所见即所得编辑环境；
4. 拥有访问权限控制；
5. 支持多种扩展方式: 宏、插件、预处理、语法着色、XML RPC……
6. 为数众多的插件中包括 TeX 科技文本输入，Media:FreeMind 思维图谱，GraphViz 示意图，gnuplot 数据图表绘图等；
7. 支持几种很实用的不同页面样式：设定方式；
8. 使用 Python 编写，真正跨平台；
9. 个人版 MoinMoin 不需要服务器就可以直接作为小型 Wiki 站点使用。

应用

本书撰写工程使用的啄木鸟社区维基，就是 MoinMoin 支持的。

维基入口：*http://wiki.woodpecker.org.cn/moin/ObpLovelyPython*

精巧地址：*http://bit.ly/2QA425*

独立版本的 MoinMoin 安装步骤如下：

1. 首先确认本地有 Python 环境；
2. 下载 MoinMoin；

 官方地址：*http://moinmo.in/MoinMoinDownload*

 精巧地址：*http://bit.ly/2uFwpK*
3. 和正常的 Python 模块安装一样，解开压缩后，进入，运行 `python setup.py install` 就可以了。

下面部署一个 MoinMoin 实例。

一般在`/usr/local/share/moin` 会自动初始化一个实例，不过一般都喜欢自己定制，安置

到习惯的目录中，也可以自行从安装包中摄取以下目录组织到任何地方

```
+--MyMoin
|  +--data      初始化的默认页面目录
|  +--htdocs    要和 Web 服务器配合的样式发布目录
|  +--server    实际服务运行文件收集，因为现在 MoinMoin 不仅仅只会 cgi 了
\  \--underlay  系统文章收集目录
```

启动一个桌面个人独立运行版的 MoinMoin，就是自个儿运行，不依赖其他服务器的维基！复制一份原始的 `wikiconfig.py` 到 server 目录中，修订 3 行：

```
,,,
data_dir = '../data/'
data_underlay_dir = '../underlay/'
url_prefix = '/wiki'
,,,
```

然后在 server 目录中运行`./moin start` 就可以在 http://localhost:8000/ 中访问到一个新的 MoinMoin 维基系统了！

详细安装说明请看网址：*http://moinmo.in/HelpOnInstalling/BasicInstallation*（精巧地址：*http://bit.ly/1pBOQR*）

美妙的功能

MoinMoin 系统提供了丰富的内容管理和发布的支持功能，其中最美妙的就是页面版本历史了。

MoinMoin 自动地将所有页面的所有修订历史都记录成页面版本（甚至于删除页面操作！），读者（当然同时也是作者）可以任意浏览任何版本的历史，决定是否回退，而且可以任意对比任何版本间的内容差异！图 PCS303-1 和图 PCS303-2 就显示了 MoinMoin 的页面版本差异对比输出。



图 PCS303-1　MoinMoin 的版本历史页面

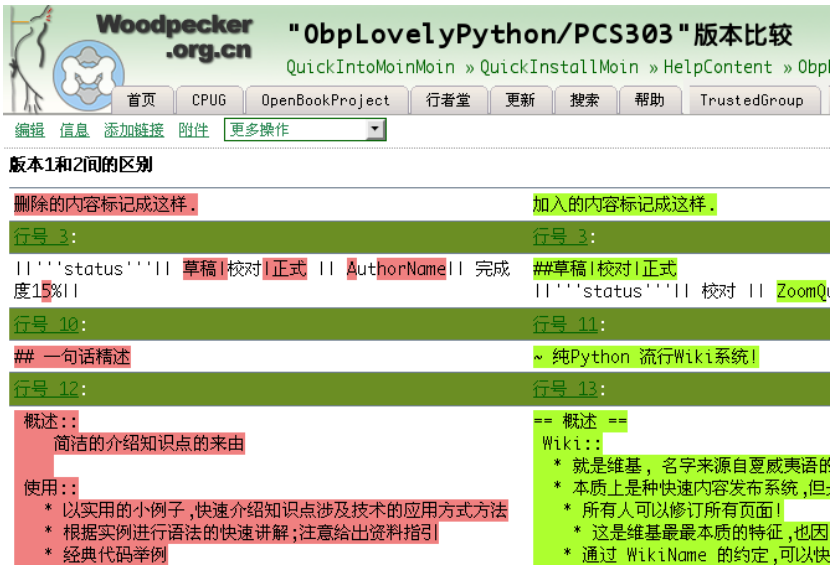


图 PCS303-2

探讨

MoinMoin 秉承优秀作品的特性，组织有非常丰富和完善的接口，所以，插件/扩展/样式极其丰富。

XmlRpcMarket：远程控制接口。

访问地址：*http://moinmo.in/XmlRpcMarket*

精巧地址：*http://bit.ly/1P1pxy*

ThemeMarket：样式收集。

访问地址：*http://moinmo.in/ThemeMarket*

精巧地址：*http://bit.ly/P7n4Z*

FormatterMarket：格式支持收集。

访问地址：*http://moinmo.in/FormatterMarket*

精巧地址：*http://bit.ly/7ufbL*

ActionMarket：功能扩展收集，支持 MoinMoin 各种操作行为增补。

访问地址：*http://moinmo.in/ActionMarket*

精巧地址：*http://bit.ly/2T2vpW*

ParserMarket：生成器收集，支持 MoinMoin 各种页面解析输出增补。

访问地址：*http://moinmo.in/ParserMarket*

精巧地址：*http://bit.ly/1gFUj3*

MacroMarket：宏收集，支持 MoinMoin 各种正文自动化处理增补。

访问地址：*http://moinmo.in/MacroMarket*

精巧地址：*http://bit.ly/34ANV7*

小结

MoinMoin 系统因为不用数据库来支撑，而且使用 Python 作为核心动力从而天然具有以下妙处：

1. 迁移/复制/备份/恢复，异常方便！因为系统全部信息都是文件；
2. 非常容易扩展！ 因为 Python；
3. 非常容易复用其中的部分功能到另外系统中！ 因为 Python ；
4. 非常容易进行定制！因为 Python；
5. 非常容易部署！因为 Python，MoinMoin 有 N 多种运行方式，可以快速部署到任何系统中~只要有 Python！

想运用维基系统进行知识管理，信息发布，团队协作等等信息快速管理的读者，推荐体验一下 MoinMoin 吧！

PCS304　Python Web 应用框架纵论

Web 是互联网的泛称，Web 应用是基于互联网的应用的简称，框架（Framework）则是整个或部分系统的可重用设计，表现为一组抽象构件及构件实例间交互的方法。另一种定义认为，框架是可被应用开发者定制的应用骨架（前者是从应用方面，而后者是从目的方面给出的定义）。

- 框架是可复用的设计构件库；
- 框架就是某种应用的半成品；
- 框架是一个可复用设计，由一组抽象类及其实例间协作关系来表达；
- 框架是在一个给定的问题领域内，一个应用程序的一部分设计与实现；
- 框架是为解决代码复用问题的一个最靠谱的尝试；

……

Python Web 应用框架可理解为由 Python 实现的面向互联网应用开发的框架！对于开发者来说，就是一堆模块和使用这些模块的文档，基于这些框架，可以快速完成有很高品质的互联网应用系统，而不用要求开发者有多么高深的相关领域经验。可以说，框架就是游戏里的组合技术，掌握了就立即可以从菜鸟变成高手！不过，在 Python 世界里，仅仅针对互联网应用的专门框架，就有不下 20 种！以至有人专门研究“为何有如此多的 Python Web 实用框架”。

访问地址：*http://xlp223.yculblog.com/post.1634226.html*

精巧地址：*http://tinyurl.com/6yajuo*

“为何有如此多的 Python Web 实用框架”的原文是“Why so many Python web

frameworks?”

访问地址：*http://bitworking.org/news/Why_so_many_Python_web_frameworks*

精巧地址：*http://tinyurl.com/6kv9j3*

在文章中，Joe Gregorio 为了展示任何人可以通过 Python 快速创建自个儿的 Web 应用框架，当场使用 8 个文件（6 个 Python 脚本，一个页面模板文件，一个服务器脚本）创建并运行了一个含有足够功能的应用框架！ 以下内容回答了“为何有如此多的 Python Web 实用框架”：因为**实现一个忒简单了**！

接下来，允许笔者将此框架整体展示一下。

Joe Gregorio 的超级框架

组成

1. model.py（数据库设计模板脚本）

```
1 from sqlalchemy import Table, Column, String
2 import dbconfig
3
4 entry_table = Table('entry', dbconfig.metadata,
5           Column('id', String(100), primary_key=True),
6           Column('title', String(100)),
7           Column('content', String(30000)),
8           Column('updated', String(20), index=True)
9        )
```

2. dbconfig.py（数据库连接配置脚本）

```
1 from sqlalchemy import *
2 metadata = BoundMetaData('sqlite:///tutorial.db')
```

3. manage.py（服务管理脚本）

```
1 import os, sys
2
3 def create():
4    from sqlalchemy import Table
5    import model
6    for (name, table) in vars(model).iteritems():
7       if isinstance(table, Table):
```

```
            table.create()

def run():
    '''使用 WSGI 模式启动服务
    '''
    import urls
    if os.environ.get("REQUEST_METHOD", ""):
        from wsgiref.handlers import BaseCGIHandler
        BaseCGIHandler(sys.stdin, sys.stdout, sys.stderr,
        os.environ).run(urls.urls)
    else:
        from wsgiref.simple_server import WSGIServer, WSGIRequestHandler
        httpd = WSGIServer(('', 8080), WSGIRequestHandler)
        httpd.set_app(urls.urls)
        print "Serving HTTP on %s port %s ..." % httpd.socket.getsockname()
        httpd.serve_forever()

if __name__ == "__main__":
   if 'create' in sys.argv:
        create()
   if 'run' in sys.argv:
        run()
```

4. main.cgi（服务器运行脚本）

```
#! /usr/bin/python2.4
import manage
manage.run()
```

5. urls.py（基于 URL 的对象选择器）

```
import selector
import view

urls = selector.Selector()
urls.add('/blog/', GET=view.list)
urls.add('/blog/{id}/', GET=view.member_get)
urls.add('/blog/;create_form', POST=view.create, GET=view.list)
urls.add('/blog/{id}/;edit_form', GET=view.member_get,
 POST=view.member_update)
```

6. view.py（基于 WSGI 应用的多个视图）

```
import robaccia
import model

```

```
def list(environ, start_response):
    rows = model.entry_table.select().execute()
    return robaccia.render(start_response, 'list.html', locals())

def member_get(environ, start_response):
    id = environ['selector.vars']['id']
    row = model.entry_table.select(model.entry_table.c.id==id).
    execute().fetchone()
    return robaccia.render(start_response, 'entry.html', locals())

def create(environ, start_response):
    pass
def create_form(environ, start_response):
    pass
def member_edit_form(environ, start_response):
    pass
def member_update(environ, start_response):
    pass
```

7. robaccia.py（模板处置脚本）

```
import kid
import os

extensions = {
    'html': 'text/html',
    'atom': 'application/atom+xml'
}

def render(start_response, template_file, vars):
    ext = template_file.rsplit(".")
    contenttype = "text/html"
    if len(ext) > 1 and (ext[1] in extensions):
        contenttype = extensions[ext[1]]

    template = kid.Template(file=os.path.join('templates',
    template_file), **vars)
    body = template.serialize(encoding='utf-8')

    start_response("200 OK", [('Content-Type', contenttype)])
    return [body]
```

8. list.html（页面应用模板）

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<html xmlns:py="http://purl.org/kid/ns#>">
<head>
 <title>A Robaccia Blog</title>
 </head>
<div py:for="row in rows.fetchall()">
<h2>${row.title}</h2>
<div>${row.content}</div>
<p><a href="./${row.id}/">${row.updated}</a></p>
</div>
</html>
```

应用

1. 创建数据库

```
~$ python manage.py create
```

2. 通过交互环境，初始化数据

```
~$ python
Python 2.4.3 (#2, Apr 27 2006, 14:43:58)
[GCC 4.0.3 (Ubuntu 4.0.3-1ubuntu5)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import model
>>> i = model.entry_table.insert()
>>> i.execute(id='first-post', title="Some Title", content="Some pithy
text...",
  updated="2006-09-01T01:00:00Z")

>>> i.execute(id='second-post', title="Moving On", content="Some not so
pithy words...",
  updated="2006-09-01T01:01:00Z")
```

3. 独立运行

```
~$ python manage.py run
Serving HTTP on 0.0.0.0 port 8080 ...
```

也可以部署前述 main.cgi 到各种 Web 服务器中的运行、发布、应用。

能力

- 通过 SQLAlchemy 进行高效对象化的数据库操作。

SQLAlchemy 官方网站：*http://www.sqlalchemy.org/*

- 使用 WSGI（Web 服务器网关接口）模式运行。

 WSGI 官方网站：*http://www.wsgi.org/wsgi/What_is_WSGI*

 精巧地址：*http://bit.ly/3IkgU4*

- 可对 URL 进行映射管理。
- 使用 Kid 模板系统，高效输出数据。

 Kid 官方网站：*http://www.kid-templating.org/*

 精巧地址：*http://bit.ly/2YDxPd*

由此可见，Joe Gregario 超级框架是一个完备的、全功能 Web 应用框架！通过极少的配置和开发，就可以完成一个动态数据展示页面！

现状

正是因为使用 Python 能够轻易地将各种既有模块组合出一个 Web 应用框架来，所以，不像其他语言世界仅有唯一或是极少数 Web 应用框架。在 Python 世界，有太多太多的应用框架，以致于大家被选择哪个深深困惑着！既怕选错框架，带来开发的麻烦；又怕一旦选定了框架，将来享受不到其他框架更加美妙的特性，患得患失中，痛苦度日。这里行者根据一些使用过的框架，综合考虑各种因素，大致给出比较靠谱的 Python Web 应用框架理解来，以便帮助大家面对异常丰富的 Python Web 应用框架，如何保持“平常心”，正确对待各种框架。

回顾

按照笔者接触到框架的时间顺序来排列，大致是这样的。

1. Zope
2. Twisted
3. Snakelets
4. CherryPy
5. Quixote
6. Karrigell
7. WukooPy
8. web.py
9. Django
10. TurboGears
11. Pylons
12. Eurasia
13. Web2py
14. Uliweb

这里没有给出详细的开发社区情况，以及笔者体验框架的详细日期和情景，只是大致地列出先后接触/了解/应用框架的顺序。以此为基础，分享一点对 Python Web 应用框架的体验。

- 各个框架都有鲜明的个性！ 关注的问题领域各不相同；
- 各个框架都有明确的社区，有专门的开发团队来维护（即使仅仅是一个开发人员）；
- 各个框架都能在不同程度上和其他框架/模块连接使用（这是 Python 的胶水特质决定的）；
- 对各个框架的模板理解各不相同，甚至于有专门研究模板系统的社区！

模板系统

在 Web 应用开发中，Template（模板）是为了解决应用逻辑和网页表现间的矛盾而产生的！这一矛盾的根本原因在于：应用逻辑是在脚本代码中制定的，是必须程序化的、动态化的；而网页表现是在 HTML 代码中定义的，是要求文本化的、静态化的！而且，应用逻辑和页面表现分离，也有利于设计和开发团队并行，提高网站的开发效率！所以，模板系统就是一种解析器，可以将类似 HTML 代码的文本和应用逻辑代码结合，输出含有动态数据的静态 HTML 页面！

在 Python 中，其实是有内置模板支持的。

访问地址：*http://jtauber.com/2006/05/templates.html*

精巧地址：*http://bit.ly/QMrd*

但是，这种支持是非常简单的，在一个复杂的功能网站开发中：

1. 设计团队期望模板系统支持的模板和 HTML 是非常接近的，是可以直接在浏览器中观察到效果的（即使没有动态数据嵌入）；
2. 开发团队期望模板系统完成以下基础任务。
 - 可以从模板文件中理解/辨别出显示用的 HTML 代码；
 - 可以将模板文件和实际生成的数据结合，生成输出页面；
 - 允许同时处理多个模板；
 - 允许模板的嵌套；
 - 允许对模板中的某个单独的部分进行处理；
 - 用起来要简单。

所以，前端模板系统也逐渐成为一个独立的领域，流行模板汇总的网址是：*http://wiki.woodpecker.org.cn/moin/PyTemplates*（精巧地址：*http://bit.ly/11OWO*）

Python Web 应用框架的模板系统一般分两类。

一则“编译型”，事先编译成可执行中间代码，输出时可以直接融合入数据；这类模板一般是从其他语言继承过来的，一般使用 XML 或是其他独特的中间数据结构声明文本，以便和 HTML 的样式模板结合进行编译；

二则“解释型”，需要动态加载 Python 运行渲染出结果再输出；这类模板通常允许嵌入 Python 语句段，可以更加灵活地组合页面输出；

伴随 Ajax 技术发展出现的第三种模板“JS 模板”——完全由 JScript 脚本在客户端动态绘制出来！ 这方面暂时没有成熟的模板系统可以直接和 Web 应用框架结合，要人工进行开发和调试，暂时不推荐使用。

分类

根据前述“Joe Gregorio 的超级框架”，可以进一步地明确一个功能完备的 Web 应用框架，至少应该包含以下方面的功能支持：

1. 服务发布
2. URL 分发
3. 模板支持
4. 数据库处理

各种框架通常对前 3 个方面都支持得不错，所以我们可以大致根据各框架对第 4 个功能的支持情况将所有 Python Web 应用框架分成以下几类。

平台性框架

这类框架认为框架不应该依赖数据库，而应关注服务的高效构建和运行平台的稳固；Zope 和 Twisted 属于这类，都是非常深邃的框架！

Zope 发布年代之早，开发时间之长，造成的影响之大，甚至于有人认为“未来互联网就是 Zope”；开创了一系列 Web 应用开发的新技术和理念，形成了宏大的社区和模块树；虽然学习曲线实在太陡，但是过来人都说“只要是作 Web 应用开发，学到底就会发现，其

实一切都在 Zope 中实现过了！”

Twisted 也是老牌社区的作品，关注网络应用底层支持，支持几乎所有网络协议，关注企业级的网络服务构建；只是缺少立等可取的应用框架，一切都得用 Twisted 去现实，算是提供了成套的钻石级工具箱！

轻小型框架

这类框架认为数据库不是应用的必要因素，而是关注友好快速的简单任务型网站开发的；CherryPy、Quixote、Karrigell、web.py 等都属于这类框架，标志特性是没有内置的数据库（类似 MySQL 的关系型主流数据库）支持模块。突出的共同特色有：

- 配置简便；
- 模板系统简单轻便，或是可以自由使用外部模板系统；
- 调试相对方便，有的甚至有专门的问题回溯机制。

一站式框架

这类框架认为数据库是应用的重要因素，关注如何仅基于本身就可以快速独立地实现一个功能型网站；Django、TurboGears、Eurasia、UliWeb 等都属于这类框架。突出的共同特色有：

- 有内置的 ORM 模块支持数据库的对象化操作；
- 有内置的事务性功能支持（比如说登录认证）；
- 有高级的模板系统，支持复杂的页面组合，有的甚至有内置的 Ajax 页面动态效果支持。

根据关键功能组件的实现，又分两类：

一则“All in one 式”：任何方面的功能都是自行开发内置包含的；胜在内部契合严密，所以运行效率比较好；但是，一旦有问题就非常难以清查，而且一般很难做到平滑的版本兼容；吻合框架针对的领域开发起来比较直接，但是难以定制支持另外的情景，Django 是其中之翘楚。

二则“Mix-in 式”：多数方面的功能是直接使用第三方模块来完成的；胜在选择自由，可以使用各种类似的已熟悉的功能组件来替代默认的，学习成本小；而且各个组件可以单独升级，版本兼容危机小；但是，由于要兼顾各种组件接口，隐患要多些，又是通过中

间层来进行配合的，数据通过了很多类转发，运行效能可能比较低；TurboGears 是其中的经典作品。

选择

现在，虽然可以快速理解现在所有的 Py 框架，但是谁也无法保证不再出现新的好框架；那么，如何理性地、靠谱地为自个儿的应用选择一个合适的框架？

笔者推荐一个思路，根据项目的特征来进行评定、取舍！核心问题有：

A. 项目是否需要长期运营？

B. 项目是否足够复杂多变？

判决矩阵:

A	B	
问题状态		**框架选择**
Y	Y	平台型框架最佳
Y	N	轻小型框架即可，平台型框架最佳
N	Y	一站式架最佳
N	N	轻小型框架最佳

以上是综合考虑以下因素后总结的决策矩阵：

1. 长期运营时，关键成本在维护，这要求框架足够稳定和容易理解；
2. 业务复杂时，关键成本在开发，这要求框架有足够的内置支持，以便减少开发投入；
3. 平台型框架都有浑厚的积累，可以满足一切要求，但是需要一定的学习成本，以便恰当地运用好这些特性；
4. 轻小型框架由于代码精悍，一般都有很好的负载能力，而且代码简洁，容易定制和扩展；
5. 前端模板系统的选择主要考虑编译型模板运行效率要高过解释型，但是开发时，调试方面则是解释型的比编译型的更直观方便；
6. 功能网站的运行效率，很大程度和模板效率及数据 I/O 处理效率相关！

框架不是最重要的

世事无常，虽然我们反对“手中有锤子，就将世界看作是由钉子组成的!”但是，在 Python 世界中，咱们这“锤子”可是金箍棒的种！尽管选择对口的框架可以事半功倍，毕竟框架是那一群牛人在总结了他们的开发经验后提炼出的；而且，自个儿工作领域不可能百分之百地刚好和那群牛人面对的问题域相同；所以，更多时候，用熟悉的框架，比用流行的框架要靠谱！因为 Python 本质特性支持我们快速扩展模块，使之成为全新的工具！UliWeb/Eurasia，以及久远之前停止开发的 WukooPy（悟空智能）这些原创框架，都在厌烦增补他人的框架后，产生了全新框架，也都可以解决一定领域中的需求；在 Python 世界，框架从来不是最重要的，最重要的应该是我们使用 Python 的决心和 Pythonic 带来的愉悦感，它促使我们可以高效地再生出更加好的工具来!

导读

基于以上对 Python Web 应用框架的体验，笔者精选出非常有代表性的框架，特邀专家分别撰写了技术文章，在此，笔者满怀敬意地先进行小小的导读。

Zope（超浑厚框架）

由中华 Zope 推广第一人 Jungyong Pan（潘俊勇）亲笔撰写！老 Pan 对 Zope 技术之痴迷到了职业水平，为推广 Zope 技术组建了“润普科技”（*http://zopen.cn*），专职进行基于 Zope 的企业信息化解决方案的实施，而且创建了 CZUG——中国 Zope 用户组（*http://czug.org*），翻译/整理了大量的 Zope 相关技术文档，为我们学习使用 Zope 平台提供了丰厚的资源!

在这篇文章中，可以学习到 Zope——这一从开始就冲着改造世界这一宏伟目标去的技术平台——的基础思想和发展路线，从而树立信心，敢于跳入这一深海中。

Quxiote（豆瓣动力核心）

豆瓣网（*http://www.douban.com*）已经成为全球流量最高的纯 Python 实现网站！ 而使用的应用框架 Quxiote 却是个轻小型框架，豆瓣团队是怎么做到的呢？！QiangningHong（洪强宁）是豆瓣核心开发成员，由他撰写的介绍文章，从实际的开发体验出发，道出了 Quxiote 堂吉诃德这一个性框架的本质思想和运用技法，非常难得!

Eurasia（关注高性能的原创框架）

沈崴，人称沈游侠，是因为那篇雅俗共赏的妙文“Python 历史书·GUI 部”。

访问地址：*http://wiki.woodpecker.org.cn/moin/PyHiStory/PyGuiHistoric*

精巧地址：*http://bit.ly/3R8nHK*

事实上沈崴作为大型 IT 公司的资深架构师，一直在亲手打造适合自身的应用框架！ 仅仅从 Eruasia3 的内部名称 `Genhiskhan`（成吉思汗）再结合本身的名称 Eurasia（欧亚），就可以知道是有多么的自信和自豪了！这一切是源自 StacklessPython 的变态能力，以及开发团队变态丰富的项目经验，再加上沈游侠变态的架构能力，最终将这一“恶魔级别”的框架带到了人间，大家可以从这篇流畅的短文中体验到 `Eurasia` 这一源自真实的大规模应用需求的高级框架的思想和个性。

UliWeb（关注平衡感的自制框架）

这又是一个中国人原创的应用框架，作者是 Limodou（李迎辉），实际上笔者体验过的各种框架，都是在 Limodou 的带领下逐一进行的； Limodou 长期独自在业余时间进行 Python 的学习和开发，先后完成了不少作品，其中最成功和知名的就是 UliPad（*http://ulipad.appspot.com*），这是个纯 Python 实现的编辑器，支持各种高级的辅助编程功能。正是在丰富的开发和应用经验基础上，Limodou 逐渐发现了现有的所有 Python Web 应用框架都不是完美的，都有这样那样的缺点。所以，Limodou 决定自个儿写出满足自己所有期望的好框架来，这就是 UliWeb 的由来！ 而且 UliWeb 很大程度上应用了很多在桌面编辑器软件 UliPad 中的开发经验，提供很多命令行的框架应用操控支持，这是其他框架所没有的特性；虽然现在 UliWeb 还在开发中，但是已经可用，而且形成了独特的应用世界观，大家可以亲自从框架作者的简介中去感受这一世界观。

Zope 史话

关于作者

潘俊勇，易度网（*http://everydo.com*）首席架构师，CZUG（*http://czug.org*）站长，《Plone 定制开发中文指南》主要作者，Zope 技术在国内的主要推广者之一。

什么是 Zope

现在很流行轻量级开发框架。在 Python 社区，就有 Django、Pylons、Quxiote 等框架，简

单易学好上手。做一个网站，费不了多少功夫就可以完成了，而且集成了很多最新的 Ajax 效果。

但是在某些情况下，我们需要面对更复杂的应用，需要考虑可重用性，需要组织大规模的开发。这时候，这些轻量级框架，可能就存在一些瓶颈了。比如企业级关键业务系统，比如银行交易等。在 Java 的世界里面，这些被认为是 J2EE 的专有领地，虽然有很多 Java 人不喜欢 J2EE 的过于复杂。

在 Python 的世界里面，是否有类似 J2EE 的企业开发框架？如果有，在以简洁漂亮著称的 Python 世界里面，他是否也会如同 J2EE 般的复杂？嗯，我来告诉你：有的，她就是 Zope（*http://zope.org*），一个 Python 上的应用服务器。她比轻量级的 Web 开发框架来得厚重，但远比 J2EE 开发简单。

谁在用 Zope?

Zope 由来已久，早在 1996 年就出现了，因此用户满天下。包括 GE、美国海军、波士顿在线等，都在用 Zope。

开源内容管理系统 Plone（*http://plone.org*）是基于 Zope 开发的，Plone 被很多世界 500 强的公司以及各国政府所使用，包括美国中情局以及 Novell 等。

Ubuntu 的社区开发站点 launchpad（*http://launchpad.net*），也是基于 Zope 开发的。launchpad 类似 code.google.com，或者 sf.net 这样的定位。

在国内，易度互联网在线工作平台（*http://everydo.com*）就是基于 Zope 开发的。这个平台提供了一组集成个人、部门、工作组、产品和项目工作管理的套件产品，让普通企业利用互联网作为基本工作平台成为可能。

Zope：对象发布环境

Zope 的完整名字是 Z Object Publishing Environment，也就是 Z 对象发布环境，这里的 Z 没有什么特别的意义。这个对象发布环境，是 Zope 的一个关键特性。

我们来看看一个典型的路径：

http://zopen.cn/products/dms（精巧地址：*http://bit.ly/3CZazP*）

传统的 Web 服务应用会有一个控制器，将上面的 URL 路径映射到实际的功能模块。这个

是以功能为中心的。

但是 Zope 是完全以对象为中心的，也就是说路径中的 products、project 都是一个个对象，Zope 自身负责对象的定位查找和调用。这个过程，也就是所谓的对象发布过程。

对象发布，是传统 Web 开发的一个大的变革，她更简单直接。特别是对于内容管理系统，由于内容之间一般存在明显包含关系，这种对象发布模型尤其适合。

ZODB：对象数据库

对象发布环境中，一切都是对象。那么对象如何存储呢？

如果把对象存放到关系数据库中，一定需要一个 O-R 映射过程。虽然有很多工具来自动化这个适配过程，但是仍然是存在阻抗的。

Zope 自带了一个对象数据库 ZODB（Z Object Database），允许你直接把对象存放到数据库中。使用对象数据库，真正实现了对象数据存取的零阻抗，你根本不会感觉到数据存取的过程！

ZODB 也发展了 10 年多的时间，现在已经非常稳定，已经很成熟了。ZODB 可以脱离 Zope 独立使用。

对象发布，配合 ZODB，让开发过程异常简单。

ZCA：组件架构

Zope 自身是一个复杂的应用。Zope 上的其他应用，比如 Plone，更加复杂。

面对复杂应用，我们须要让系统扁平化，减少依赖，降低耦合，提升代码的可重用性。所有这些内容，也是设计模式（Design Pattern）所关注的课题。组件开发，正是应对这一需求的应对技术。微软的 COM、Mozilla 的 xpcom，都是著名的组件开发框架。

Zope 也提供了一个组件开发框架 ZCA（Zope Component Architecture），他为 Python 引入了接口的概念，同时提供了接口注册和查询的机制，使得基于接口而不是实现编程成为可能。

传统的开发是直接基于类开发，这样，和具体的实现就紧密耦合了。接口是功能的契约，实现同一接口的类可能有很多。通过接口开发更加灵活。

在 ZCA 中，最基本的组件是适配器组件。大部分对象之间的关系，都可以用适配器来描

述。在现实生活中，比如显卡就是从 PCI 接口到 VGA 接口的适配，电源就是 220V 交流到 9V 直流的适配，投影仪就是从 200V 电源和适配信号到投影光信号之间的多适配。基本上可以说，一切皆适配。这也是 ZCA 的核心理念。

在适配组件的基础上，ZCA 还衍生出工具组件、事件订阅组件等高级组件。这些组件都可以通过 XML 文件来装配，构成最终的应用。

ZCA 是实践设计模式的最佳技术手段。使用 ZCA，你根本不需要太多思考，便可解决依赖问题，你可让设计模式成为一种习惯和标准。同时也让经过设计模式处理过的代码不再怪异，让代码的阅读和维护更加轻松。

ZCA 是 Zope 总结经验教训的一个产物，也是 Zope 最有价值的产品之一。ZCA 可以脱离 Zope 使用。ZCA 其实更应该说是 Python 的组件开发框架，现在也已经被 twisted 等项目采纳。

由于 Zope 内核采用 ZCA 开发，Zope 更加可扩展，这也是 Zope 适合企业级开发的一个原因。

你可从以下网址查看 ZCA 的文档：*http://docs.everydo.com/zope3/ca*（精巧地址：*http://bit.ly/1UCdvX*）

Repoze：让 Zope 融入 Python 世界

我们一直在说 Zope 厚重。这种厚重，让 Zope 在 Python 世界里很另类。很多习惯了简单的 python 开发人员，不大爱 Zope 的这种一眼望不见底的深度。

Zope 的很多的特性，包括认证、对象发布、事务管理、授权等，功能都很强大，但是几乎无法在 Zope 世界外使用，这样 Zope 世界显得有些封闭。

这种现象，其实也不是 Zope 独有的。Python 上大量的 Web 框架，大都是各自为政，彼此互通的很少。

WSGI 是解决这一问题的途径。WSGI（Web Service Gateway Interface）定义了 Web 服务器和 Web 应用及 Web 中间件之间的交互协议。这样，只要支持 WSGI，那么各种 Web 服务器、Web 应用和中间件，就能相互对接了。比如，你可轻松让你的网站 wiki 采用 MoinMoin，而发布系统采用 Plone。

而 Repoze（*http://repoze.org*）做了什么呢？Repoze 是一个“拆卸工”，他把复杂/强大的 Zope，逐一分解成一个个 WGSI 组件。这样，Zope 基本消失了，Zope 的强大特性，可以被 Zope 外的各种框架所使用。

目前，Zope 的可插拔认证系统、Zope 的事务管理、对象发布，均被 Repoze 给 WSGI 化重写了。Zope 坚硬的外壳，已经被 Repoze 敲开，营养已经被 Repoze 所吸收，Repoze 太“狠”了!

Repoze 又推出了自己的开发框架 repoze.bfg（*http://static.repoze.org/bfgdocs/*，精巧地址：*http://bit.ly/1Sz2Ou*），这个是利用了 ZCA 的一个可以一眼见底的“轻量级”开发框架，和 pylons 和 Django 有神似的地方。

Repoze.bfg 实际上是 Zope 的一个分支，Repoze.bfg 未来非常值得期待。虽然现在还处在早期，但是早已有蜻蜓落上头，已经有很多应用基于 Repoze.bfg 开发了。我相信，Repoze 是 Zope 的终极出路，是众望所归。

Zope 曲折发展史

Zope 发展其实有一段曲折的历程，如同 Z 字的形状，一波三折。

Zope 第一个辉煌，是突破了传统 CGI 编程的复杂性，推出了直接通过浏览器进行的脚本开发，这大大简化了 Web 开发过程。然而脚本开发存在代码管理不方便的问题，很多开发设计模式无法用上。大量轻量级开发框架的普及，逐步淡化了这一特性。但是目前在 Plone 中，直接通过浏览器进行定制，仍然是 Plone 的关键特性之一。

Zope 第二个辉煌，应该是 Plone 内容管理系统的流行，Plone 的流行，让 Zope 的用户和开发人员迅速扩展。Plone 是基于 Zope 内容管理框架 CMF 开发的。

Zope 第三个辉煌，应该是 Zope3 的推出。Zope3 是对 Zope 从前版本的重写，组件架构 ZCA 就是在这个版本中引入的。但是由于 Zope3 面对的是复杂应用，这种曲高导致了和寡。Zope 在整个 Python 社区并不十分流行。特别是在现在 ROR/Django 势强，而 J2EE 势微的年头。

现在正在走入第四个辉煌，那就是 Zope3 的轻量级化，让 Zope3 成为每个开发人员的挚爱。Grok（*http://grok.zope.org*）就是一个尝试，他模仿了 ROR 的很多概念，开发过程简单很多，不再要配置 XML 文件。前面提到的 repoze.bfg，则是另外一个尝试，bfg 并不忌讳 XML，因为他的配置 XML 很简单，我个人更看好 bfg 的前途。

如何上手？

Zope 的世界太庞大，你困惑了吗？嗯，如果你喜欢 Pylons 的简洁，是一个思维严谨的开

发人员，你希望一切都可操控，那建议你选择 repoze.bfg，他会让你满足。

如果你在开发一个相当复杂的、相当严肃的应用，你还是选择 Zope3 吧。Zope 3 有几乎所有你想要的东西，该走的弯路，别人都走过了，你可省很多力气。

当然你需要有一个可以相互学习的团队，须要经历一定的入门门槛过程。

如果你只是想做一个网站，那选择 Plone 吧，这个是专业级别的，你只须掌握一些定制技术。

你根本不做 Web 开发？哦，那你去看看 ZODB、ZCA 吧，相信他们会对你有益的。

Quixote——豆瓣动力核心

本文作者

洪强宁，网名 Qiangning Hong 或 hongqn，2002 年开始接触 Python，2004 年起完全使用 Python 工作。豆瓣网（*http://www.douban.com*）02 号程序员，技术负责人。

缘起

Quixote 是由美国全国研究创新联合会（CNRI，Corporation for National Research Initiatives）的工程师 A.M.Kuchling、Neil Schemenauer 和 Greg Ward 开发的一个轻量级 Web 框架。和几乎所有的开源项目一样，Quixote 也是为了满足实际需要而出世的。

CNRI 当时在进行一个名为 MEMS Exchange 的项目（*http://www.mems-exchange.org/*）。MEMS 是微机电系统的缩写，制造一个 MEMS 设备往往需要多种制造设备，单个工厂可能无法提供所需的所有设备。因此，MEMS Exchange 项目就是要整合起多家制造厂的资源，利用互联网派单和追踪制造过程，形成一个分布式的 MEMS 设备制造网络。

起初，他们做了一个 Java 版的客户端程序提供给用户，但他们发现，没有人愿意使用这个客户端程序，大家还是习惯性地用邮件发送加工过程。最终他们认识到，虽然客户端的表现力更强，功能也更完整，但相比起要下载一个庞大的程序，用户更加愿意使用他们每天面对的浏览器来做事情。于是，他们决定改到 Web 界面上来，要做一个 Web 应用。但是用 Java 的 servlets 开发 Web 应用是一件非常低效的事情，所以他们选择了 Zope（和现在不同，在 1999 年，Python 的 Web 应用框架没有什么选择的余地，基本上是 Zope 一

家独大）。3 个月的开发之后，他们得到了一个运转良好的系统。

然而，Zope 带来的快乐并没有持续多长时间。几个月后，他们想提供更加复杂一点的界面，却发现用 Zope 写的代码难以维护和调试，在浏览器的文本编辑框里写代码也实在不是什么好的体验。由于当时除了 Zope 之外也没有什么别的 Python Web 框架，他们决定：自己写一个！在 2000 年，编写一个新的 Web 框架是类似于向风车挑战一样的事情，开发团队自嘲地用堂吉诃德的名字命名这个框架：Quixote。

特性介绍

（以下使用 Quixote 1.2 版本为例进行介绍）

Quixote 有以下几个特点：

1. 简单的 URL 分发规则

所有的 Web 框架要解决的第一件事情就是：当用户输入一个 URL，应该调用哪段代码来响应用户需求呢？这就是 URL 分发。与 Django 之类的基于正则表达式匹配来实现 URL 分发的框架不同，Quixote 是基于 URL 的目录结构进行逐层查找实现的。

一个 Quixote 应用对应着一个 Python 的名字空间（通常是一个包）。URL 的每一级目录映射到一个子名字空间上（模块或者子包），URL 的最末一级映射到上一级名字空间中的一个函数上。该函数返回一个字符串，就是返回给浏览器的页面。例如，如果你的应用是放在 app 这个包下的，那么 *http://www.example.com/hello/john* 就对应 `app.hello.john(request)`。末级目录则对应于 `_q_index` 函数，如 *http://www.example.com/hello/* 对应于 `app.hello._q_index(request)`。就这么简单。

这种层级查找的结构还可以带来额外的好处：在目录层进行控制。在每一个目录层级进行查找之前，Quixote 都会先运行当前名字空间下的_q_access(request)函数。比如你需要 *http://www.example.com/settings/* 下的所有路径都只能由登录用户访问，那么只须要在 `app.settings` 这个名字空间（`app/settings.py`，或者 `app/settings/__init__.py`）中定义一个`_q_access(request)`函数，在其中检查当前用户的登录状态（cookie 可以从 `request` 对象获得），如果发现是未登录状态，抛出一个 `quixote.errors.AccessError` 异常即可。

当 Quixote 执行代码时碰到异常，它会首先检查当前名字空间下有没有 `_q_exception_handler` 函数，如果有的话，则由这个函数处理异常，返回的字符串则为出错页面。如果没有定义这个函数，或者在执行它的时候又抛出了异常，则向上一级的

名字空间查找_q_exception_handler 函数，直至找到为止。所以，我们一般只须要在 app/__init__.py 中定义_q_exception_handler 函数，就可以方便地实现定制出错页面了（当然你连这个也不定义也可以，Quixote 提供了默认的出错页面）。

2. 易于实现 RESTful 的 URL

刚才我们说 *http://www.example.com/hello/john* 对应 app.hello.john(request)，那除了 john 之外，还有成千上万的用户的名字怎么办？不能为每个用户都定义一个函数吧！传统的方法是把会变的部分用 URL 参数的形式传进来，比如 *http://www.example.com/hello/?name=bob*。但这种风格的 URL 既不好看，又不利于搜索引擎收录，要是都能像 john 一样直接成为 URL 的一部分多好啊！幸运的是，Quixote 帮助你实现了这一点。

我们只须要在 app.hello 这个名字空间中定义一个_q_lookup(request, component)函数，当 Quixote 查不到 bob 这个名字时就会调用这个函数，把需要查找的名字 bob 传给 component 参数，用这个函数返回的结果作为找到的子名字空间或函数。在我们的例子里，代码就是：

```
1 # in app/hello/__init__.py
2 def _q_lookup(request, name):
3    def hello(request):
4       return say_hello(name)
5    return hello
```

这里我们返回了一个函数（也可以用定义了__call__方法的类实例代替），这个函数将被调用以生成 HTML 页面。

3. 显式标记，拒绝魔术

Zen of Python 中有一句“Explicit is better than implicit”，Quixote 也是这个理念的贯彻者。所有可以通过 URL 访问到的函数和方法，必须在当级名字空间的_q_exports 变量中列举出来（除了用 _q_lookup 实现的动态 URL）。也就是说，要让 *http://www.example.com/hello/bob* 可以访问，必须在 app/__init__.py 中定义

```
1 _q_exports = ['hello']
```

4. 非常类似 Python 的模板语言

呃，或者说，就是 Python 语言。为了让程序员不用再学习一门模板语言，而是直接使用已经掌握的 Python 语法写模板，Quixote 的模板语言 PTL（Python Template Language，以.ptl 作为文件名后缀）的语法和 Python 一模一样（除了后面要说到的[html]标记），在执行了 quixote.enable_ptl()之后，这些.ptl 文件就可以像普通的.py 文件一样作为 Python 模块 import 进来。

但是既然是模板语言，就会有一些专门为了生成字符串的语法糖。在函数定义时，加上一个 [html] 标签就可以改变这个函数的表现，使得每执行一条语句，运行的结果如果不是 None 的话，就会以字符串的形式添加到函数的返回值里，而无须再使用 return 语句了。假设 hello.ptl 的内容如下：

```
1 def say_hello [html] (name):
2    "Hello, "
3    "<em>%s</em>!" % name
```

我们来看看 say_hello 的返回值是什么：

```
1 >>> from quixote import enable_ptl
2 >>> enable_ptl()   # 使得我们可以 import ptl 文件
3 >>> from hello import say_hello
4 >>> print say_hello("Quixote")
```

因为 say_hello 函数由两条语句组成，这两行语句的运行结果分别返回一个字符串，因此运行结果就是 `Hello, <em>Quixote</em>!`。这个设定可以大大方便模板的编写。

5. 内置的安全性支持

互联网上有很多寻找网站漏洞的攻击者，跨站脚本（XSS）是一个常见的攻击手段。这种攻击通常会在页面显示数据的地方插入一段恶意 Javascript 代码，当有人浏览这个页面的时候，就会运行这段代码。比如在 say_hello 这个例子中，恶意用户输入的名字（也就是 name 参数）是`"<script>alert('haha')</script>"`的话，在模板没有安全性处理的情况下，页面就会输出 `Hello, <em><script>alert('haha')</script></em>`，浏览这个页面的用户就会中招。幸好，PTL 帮助我们解决了这个问题，它会自动将输入的数据进行 HTML 转义，就是用`<`、`>`等 HTML 实体替换掉危险的<、>等字符，这样输出的结果就是 `Hello, <em><script>alert('haha')</script></em>`，安全了。

PTL 是怎么实现这一点的呢？实际上，打了[html]标记的 PTL 函数中的字符串常量（比如`"<em>%s</em>!"`）并不是普通的 str 对象，而会自动转成 quixote.html.htmltext 对象，这个对象的接口和 str 几乎一样，但和普通的 str 对象做操作时（比如+、%等运算），会自动把 str 对象中的 HTML 特殊字符进行转义。

6. 高效的模板

PTL 的速度非常之快，在豆瓣，我们曾经拿它和号称最快的 Python 模板 Mako 进行过 PK，PTL 胜出。当然，这一部分是由于 PTL 的核心代码是用 C 写的（比如前面说的 htmltext 类），另一方面 PTL 的功能非常基本，完全依赖于 Python 语言自身的特性。

7. 没有内置数据库支持

Quixote 专注于 URL 分发和模板系统，对于数据库没有提供直接支持。开发者可以根据自己的需要选择合适的数据库软件，如 SQLAlchemy、SQLObject 或者直接使用 MySQLdb 等。

快速起步

那我们来看一个完整的应用示例吧，就是我们前边举的例子：通过 URL 获得一个名字，在页面上显示对他的欢迎信息。

首先安装 Quixote，在 *http://quixote.ca/* 下载 Quixote 1.x 版的最新版本（目前是 1.2 版）并安装。

```
# app/__init__.py

_q_exports = ['hello']
```

```
# app/hello/__init__.py

from app.hello.hello_ui import say_hello

_q_exports = []

def _q_index(request):
    return say_hello("everyone")

def _q_lookup(request, name):
    def hello(request):
        return say_hello(name)
    return hello
```

```
# app/hello/hello_ui.ptl

def say_hello [html] (name):
    header(title="Hello")
    "Hello, "
    """<em class="name">%s</em>!""" % name
    footer()

def header [html] (title):
```

```
    """<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>%s</title>
</head>
<body>
    """ % title

def footer [html] ():
    """</body>
</html>
    """
```

以上已经创建了整个 Web 应用。下面的问题是：怎么运行呢？

有好几个方式可以用来运行 Quixote 应用，比如 mod_python、FastCGI 和 CGI 等。在豆瓣我们使用的是 SCGI（*http://www.mems-exchange.org/software/scgi/*，精巧地址：*http://bit.ly/2GMFUb*），这是 Quixote 的开发团队制作的一个简化版本的 FastCGI，使用和 Web 服务器独立的进程运行 Web 应用。著名的轻量级 Web 服务器 lighttpd（*http://www.lighttpd.net/*）直接内置支持 SCGI，下面用 lighttpd 来运行 hello 程序。

首先安装 scgi Python 包，到 *http://python.ca/scgi/* 下载最新版本并安装。

创建 SCGI 服务程序 `scgi-server.py`：

```
#!/usr/bin/env python

from scgi.quixote_handler import QuixoteHandler, main
from quixote import enable_ptl

enable_ptl()

class MyHandler(QuixoteHandler):
    root_namespace = 'app'

if __name__ == '__main__':
    main(MyHandler)
```

这个程序运行时会在默认的 4000 端口上开启一个 SCGI 服务，端口号可以使用命令行参数 `-p` 更改。更多控制参数请使用 `python scgi-server.py --help` 查看。

创建 lighttpd 的配置文件 `lighttpd.conf`：

```
server.modules = (
 "mod_scgi",
)
server.document-root = "."
server.port = 8080

scgi.server = ( "/" =>
     ( "localhost" => (
          "host" => "127.0.0.1",
          "port" => 4000,
          "check-local" => "disable",
          )
     )
)
```

这个配置让 lighttpd 监听 8080 端口，把所有请求都转发给 4000 端口上的 SCGI 服务处理，再把处理结果发给用户。

运行 SCGI 服务和 lighttpd（以 GNU/Linux 下为例）：

```
$ python scgi-server.py
$ /usr/sbin/lighttpd -f lighttpd.conf
```

访问 *http://localhost:8080/hello/bob*，就可以看到 "Hello, bob!"了

须注意的是，对代码进行修改后，要重启 SCGI 服务才能看到效果。我们可以做一个简单的重启 SCGI 服务的 shell 脚本，以简化操作：

```
#!/bin/sh

PIDFILE=/var/tmp/quixote-scgi.pid  # 默认的 PID 文件路径

kill `cat $PIDFILE`
sleep 1
python scgi-server.py
```

案例讲解

作为老牌的 Web 框架，Quixote 已经被证明了它足够支撑起相当规模的网站。除了 MEMS Exchange 之外，LWN（Linux Weekly News, *http://lwn.net/*）也是使用 Quixote 搭建的。在国内，Quixote 最著名的使用者就是豆瓣网（*http://www.douban.com/*）。

豆瓣网是一个致力于帮助用户发现自己可能感兴趣的书、电影、音乐、博客等信息的网

站。2005 年 3 月正式上线，当时 Python 社区的 web 框架屈指可数，Django、TurboGears 等新兴框架还没出现，因此选择了 Quixote 作为整个网站的基础框架。至 2008 年 10 月，豆瓣网已经发展到200万注册用户，每日1000万次动态页面点击，但仍只需要两台Quixote 应用服务器即可轻松负担，这充分说明了 Quixote 的性能和可扩展性。

小结

Quixote 是一个轻量级的框架，简单、高效，代码也十分简洁易读。用豆瓣网创始人阿北的话来说：“用 quixote 的时候你的注意力大部分在要实现的内容上，framework 不会拦在中间跳来跳去。”

但是，Quixote 毕竟是 2000 年的框架，现在看来已经略显老态。比起 Django、TurboGears 这些后起之秀来，确实存在某些劣势。大略有以下几点：

1. 调试开发不够方便，没有内置的调试用 Web 服务器；
2. 修改代码后必须重启服务才能看到效果；
3. URL 的末尾带和不带"/"会被解释到不同函数；
4. PTL 模板适合 Python 程序员编写，却非常不适合美工独立编辑；
5. 没有内置的 WSGI 支持；
6. 没有内置的用户认证系统、数据库支持等。

但 Quixote 最大的优势也在于它的资格够老，基本上已经没有 bug。在使用它时不用担心框架会出什么意想不到的问题。而且它只提供了最基本的功能，用户认证系统、数据库访问层等需要自行实现，虽然麻烦一些，但更利于根据你的应用量身定制。如果你的目标是追求稳定和性能，而非追求最新最炫的技术，Quixote 仍然是一个很好的选择。

Eurasia3——关注高性能的原创框架

作者

沈崴，高级架构师。在 Zoom.Quiet 的“胁迫”下写下此文，作为 Eurasia3 的作者，我很高兴和大家一起走进 Eurasia3 的世界，去了解 Eurasia3 的历史、特性，去感受

Eurasia3 的设计思想。

概述

Eurasia3 是高性能应用服务器，同时也是一套简洁高效的开发框架。除此之外，Eurasia3 项目还开发了对象数据库 MissileDB 和 JavaScript 框架 NJF，形成了一套完整的开发体系。Eurasia3 应用服务器及其组件基于 Python2.5，基本上采用 BSD 协议开源发布，可以用于闭源的商业应用；也有少数组件使用自由软件的授权协议，使用时必须遵守自由软件规范。最后，使用 Eurasia3 是免费的。

Eurasia 沿革

尽管 Eurasia3 的客户端框架从 1999 年就已经开始设计了，但是 Eurasia 项目正式出现是在 2004 年。当时我正致力于人工智能的研发，现在这套程序已成功发展为国内乃至世界上最大的人工智能应用之一，值得一提的是它完全是使用 Python 设计开发的。虽然最早我使用 Yacc/Lex 来设计人工智能描述语言，不过后来我们的团队很快编写了上百万行的人工智能代码，这使得原来的编译器不堪重负，为此我必须修改程序，将描述语言改成了 Python 的无栈实现。同时带有一整套 PyQT 编写的 IDE 开发环境。

然而接下来，怎样有效管理如此之多的逻辑数据便成了一大难题，为此 Plone 和 Zope3 被尝试用来搭建数据平台，管理界面也从 Tkinter 转到 Web 上来。这产生了 Eurasia 项目，它基于 Plone/Archetypes、采用 ZODB 对象数据库。Eurasia 使用 NJF 提供 AJAX 操作界面——此时 NJF 开发已近 5 年。

随后我们使用 Plone 以 Archetypes 的开发速度很快建立起庞大的知识管理、办公系统和信息网络。此后，我开始考虑将 Plone 用于外网。在当时，Plone 性能不佳是众所周知的，故一般仅用于内部系统。这和 Zope2 的实际性能严重不符，显然 Plone 复杂的页面用了太多 IO，用轻量级的界面替换即可。这样 Eurasia2 就产生了。

Eurasia2 的产生非常具有戏剧性，因为在 Eurasia2 设计好后我发现了一个几乎一模一样的东西，那就是 TurboGears。作为一个非常讨厌重复制造轮子的人，可以肯定的是如果先发现 TurboGears，我绝然不会开发 Eurasia2。然而事实却是 Eurasia2 和 TurboGears 不仅理念雷同，而且几乎是同时开始设计、同时完工，对我来说这是一个令人震惊的巧合。

Eurasia2 和 TurboGears 最大的区别是 Eurasia2 采用 Plone 作为后台数据库和管理系

统，这意味着 Eurasia2 可以直接利用 Plone 的权限和工作流，所以相对地 TurboGears 还是造了太多轮子，Eurasia2 胜出。在 Eurasia 的基础上 Eurasia2 使用了 CherryPy 等轻量级技术，把 Plone 从内网延伸到了外网，在诸如招聘系统等公网项目中得到了实用。

2006 年 9 月，经过讨论，为回馈开源社区，Eurasia 2.0.2 以 BSD 协议开源。在这一年 Ajax 仿佛是突然被发明出来一样，变成了炙手可热的东西，一下子火了。然而对 Eurasia 而言，Ajax 已是落日余晖，我清楚地知道一个新的时代即将到来。就在 Eurasia2 发布的同时，一个新的框架已经从构想逐渐走向现实。

在技术界有些人喜欢发明概念，比如 Comet，而另一种人务实行动。在商人还没有来得及提出概念的时候，Eurasia3 出现了。

Eurasia3 简介

Eurasia3 作为 Eurasia2 的下一代技术最初源于一系列关于网页游戏的尝试。我试图在没有插件和 Flash 的情况下，在浏览器上实现即时类游戏效果，并维持数万人在线。

漫游地图，把周边玩家和怪物的行动即时反馈给你——很多人认为单靠浏览器和 HTTP 协议是无法实现的，你得用 Socket。这没错，不过别忽略 HTTP 长连接，事实上它就是 TCP。目前多数框架都支持长连接，问题在于要维持数万、数十万乃至数百万人同时在线。Asynchronous 单线程循环很合适，但到了应用层过于复杂，故多数框架主要使用线程来维持连接状态。一万个长连接就需要一万个线程或进程，这显得很荒谬。

轻便线程 MicroThread 可以在单线程中模拟出成百上千万“看上去像是原生”的线程，且每秒可以完成百万级的调度，这正是我们想要的。Eurasia3 便是这种基于 Asynchronous IO 和 Stackless Python 轻便线程的框架，Eurasia3 隐藏了底层细节，对用户而言 Eurasia3 看上去和传统框架其实并没有什么区别，就像这样。

```
1 from eurasia.web import config, mainloop
2 def controller(httpfile):
3    httpfile['Content-Type'] = 'text/plain'
4    httpfile.write('hello world!')
5    httpfile.close()
6
7 config(controller=controller, port=8080)
8 mainloop()
```

Eurasia3 带有完备的 Web 开发支持，除 Response 外还包含表单读取、文件上传、POST

报文读取等接口，他们都简洁到可以一言以蔽之。

```
表单词典      = Form(httpfile)
文件句柄      = SimpleUpload(httpfile)
请求头部      = httpfile['Http-Header']
请求报文      = httpfile.read(size) / httpfile.readline(size)
```

Eurasia3 唯一需要用户自己处理的是 httpfile.path，即 URL。如何对待 URL，是个见仁见智的问题，甚至是许多框架的关键分歧。Eurasia3 采取了更为聪明灵活的策略，不下定论，把选择的权利交还给用户。不过这也让 Eurasia3 看上去似乎更像是一个底层框架。事实上 Eurasia3 项目已经模拟出了 CherryPy、Eurasia2 等框架，以后无论是基于 Pylons、Quxiote 还是 Django 等框架的程序都可以运行在 Eurasia3 上，并能在一个程序中混用多个框架。

然而最让 Eurasia3 感兴趣的还是实现一个 Plone，于是就有了 MissileDB 数据库。在 Eurasia3 中编写数据库应用就像是这个样子。

```
 1 from eurasia.shelve2 import open, Persistent, BTree
 2
 3 # 定义持久化对象 (Persistent) User, User 对象可以直接以对象形式保存在数据库中,
 4 # 不需要进行对象关系映射 (Object Relational Mapping), 这也是对象数据库的特点
 5 #
 6 class User(Persistent):
 7     def __init__(self, username, password):
 8         self.username = username
 9         self.password = password
10
11     def hello(self):
12         print 'Hello Im %s, can I make friends with you?' %self.username
13
14 db = open('test.fs', 'c')      # 创建并打开数据库 "test.fs"
15 db['user'] = BTree()          # 创建 BTree 结点 "user", 相当于在关系数据库
                                    中建表
16 obj = User('william', '*******') # 创建一个 User 对象
17 db['user']['william'] = obj    # 把 User 对象存入数据库
18 db.close()
19
20 db = open('test.fs')          # 重新打开数据库
21 db['user']['william'].hello()   # 访问数据库中的对象
```

基于 Eurasia3 能让你的应用获得数倍乃至数十倍的性能提升。但 Eurasia3 主要还是为高并发和长连接设计的，所以在这方面的支持要自然得多。下面这个例子将向你展示 Eurasia3 如何与浏览器建立长连接，并且进行长连接通信的过程。

```
# -*- coding: utf-8 -*-
html = '''\ # HTML 页面
HTTP/1.1 200 OK
Content-Type: text/html

<html>
<head>
   <title>Comet Example</title>
</head>
<body>
<script language="JavaScript">

// 待会服务器会远程调用这个函数
function message(msg)
{
   confirm(msg);
};

</script>
<!-- 建立 Comet 长连接, 借助 iframe -->
<iframe src="comet" style="display: none;"></iframe>
</body>
</html>'''

import eurasia
from eurasia.web import config, mainloop, Comet

sleep = eurasia.modules['time'].sleep

def controller(httpfile):
   # 输出普通页面 (当 URL 不是 comet 时)
   if httpfile.path[-5:] != 'comet':
      httpfile.write(html)
      return httpfile.close()
      return
   browser = Comet(httpfile)
   browser.begin()          # 开始和客户端之间的通讯
   browser.message('start')  # 使用原生 Python 代码直接调用浏览器端
                             # 名为 "message" 的 JavaScript 函数
                             # 发送消息 "start"

   sleep(2)                 # 每隔 2 秒调用一次客户端 message 函数
   for i in xrange(1, 3):
```

```
44        browser.message(i)
45        sleep(2)
46
47    browser.message('finish')
48    browser.end()           # 断开长连接
49
50 config(controller=controller,
51    port = 8080, verbose=True)
52 mainloop()
```

和多数基于事件的 Comet 实现不同，Eurasia3 可以使用原生的 Python 代码，在任何时候远程调用浏览器上的 JavaScript 函数。

Eurasia3 项目的 JavaScript 库 NJF 对 Comet 提供了完善的支持。

Eurasia3 VS Django

这是一个伪命题，Eurasia3 并不是一种和 Django，或者 Pylons、Zope 直接竞争的东西。它看上去要更底层一点，更接近于 Twisted——其实 Eurasia3 同样可以用来开发 TCP 服务器。

```
 1 from eurasia.web import config, mainloop
 2 def echo(sockfile):
 3    while True:
 4        data = sockfile.readline(1024)
 5        if data == 'quit':
 6            sockfile.close()
 7            break
 8        else:
 9            sockfile.write(data)
10
11 config(tcphandler=echo, port=8080) # 与前面不同, 这里使用 tcphandler
12 mainloop()
```

和 Web 应用一样，代码出奇地简单，你看不到任何底层细节，然而这个 echo 服务器却可以轻松支持数十万用户同时 telnet 上来。简约是 Eurasia3 的设计原则，它试图以最简单的方式提供最强大的功能。所以 Eurasia3 并不完全像它看上去那样是一个底层框架，尽管非常简洁，Eurasia3 实际上已经提供了足够多的功能。

- 完备的 Web 开发支持；
- 多地址、多端口虚拟主机支持；

- 常规请求每秒 8000 次以上的简单动态页面响应能力；
- 数十乃至数百万的同时在线支持，只要你有足够的内存；
- 长连接 Comet 应用开发支持，支持以 Python 原生代码远程调用浏览器 JavaScript；
- 无须外部模块程序便可以无修改地运行在 FastCGI 上，Eurasia3 本身就是另一种高效的 FastCGI 框架；
- 高性能 TCP 服务器开发支持；
- 核 CPU 支持。

开始使用 Eurasia3

Eurasia3 是跨平台的，在 Unix、GNU/Linux 系统上能得到最好的性能，同时你需要 Stackless Python 2.5，带有 greenlet 扩展的 Python 2.5 也是可以的。

Eurasia3 的代码托管在 *http://code.google.com/p/eurasia*（精巧地址：*http://bit.ly/3l4HFj*），你可以在那里下载到 Eurasia3 的最新版本。

同时，可以加入 Eurasia 用户组以获得关于 Eurasia3 的最新动态和技术支持。

Eurasia 用户组：*https://groups.google.com/group/eurasia-users*

精巧地址：*http://bit.ly/3oF8o8*

UliWeb

作者介绍

李迎辉，网名 Limodou。Python 是我的最爱，但我的主业不是 Python，它仍然是我的纯业余爱好。喜爱开源事业，虽然不是刻意去做，但是独立完成了大大小小的许多 Python 开源项目，还参与了许多 Python 项目。最大的项目就是 UliPad。而下面要介绍的 Uliweb 是我在 Web 框架方面的尝试，在做此之前，研究了一段时间 Django，并大力进行宣传，还编写过《Django Step by Step》教程。本文的写作一方面是应 Zoom.Quiet 的邀请，另一方面是为了向大家介绍我对 Web 开发的理解，并把我个人的研究成果介绍给大家。只要你努力，一样可以以自己的方式做出让自己满意的框架。

概述

Uliweb 是一个新的 Python Web Framework，它之所以会产生是因为现有的框架多少有些令人不满意的地方，而且许多情况下这些不满意的地方或多或少对于 Web 开发有影响，因此在经过对不少框架的学习之后，我决定开发一个新的框架，希望可以综合我认为其他框架中尽可能多的优点，同时使这个新的框架尽可能简单、易于上手和使用。不过，这个框架目前主要还是一个人在做，并且是业余在做，所以在进度上相对要慢一些。

Uliweb 按照 GPL v2 协议开放源代码。Uliweb 并不是一个从头开始的框架，它使用了一些较为成熟的库，如：用来进行命令行、URL 映射、Debug 等核心处理的 Werkzeug；用来生成和处理请求、响应对象的 webob；强大的 ORM 库 SqlAlchemy 等。Uliweb 在开发中还借鉴了像 web2py 的 Template 模板模块、Django 的一些设计思想和成果。

功能特点

组织管理

- 采用 MVT 模型开发。
- 分散开发统一管理。采用 App 方式的项目组织。每个 App 有自己的配置文件、templates 目录、static 目录，这使得 Uliweb 的 App 重用非常方便。同时在使用上却可以将所有 App 看成一个整体，可以相互引用静态文件和模板。默认所有 App 都是生效的，也可以指定哪些 App 是生效的。所有生效 App 的配置文件在启动时会统一进行处理，最终合成一个完整的配置视图。

URL 处理

- 灵活强大的 URL 映射。采用 Werkzeug 的 Routing 模块，可以非常方便地定义 URL，并与 View 函式进行绑定。同时可以根据 view 函式反向生成 URL。支持 URL 参数定义，支持默认 URL 定义，如：

```
appname/view_module/function_name
```

View 与 Template

- view 模板的自动套用。当 view 返回 dict 对象时，自动根据 view 函式的名字查找对应的模板。
- 环境方式运行。每个 view 函式在运行时会处于一个环境下，因此你不必写许多的 import，许多对象可以直接使用，比如 request、response 等。可以大大减少代码量。
- 模板中可以直接嵌入 Python 代码，不需要考虑缩进，只要在块结束时使用 pass。支持模板的 include 和继承。

ORM

- 支持 Model 与数据库的简单自动迁移，包括自动建表和表结构的修改。

i18n

- 支持代码和模板中的 i18n 处理。
- 支持浏览器语言和 cookie 的自动选择，动态切换语言。
- 提供命令行工具可以自动提取 po 文件，可以以 App 为单位或整个项目为单位。并在处理时自动将所有语言文件进行合并处理。当发生修改时，再次提取可以自动进行合并。

扩展

- plugin 扩展。这是一种插件处理机制。Uliweb 已经预设了一些调用点，这些调用点会在特殊的地方被执行。你可以针对这些调用点编写相应的处理，并将其放在 settings.py 中，当 Uliweb 在启动时会自动对其进行采集，当程序运行到调用点位置时，自动调用对应的插件函式。
- middleware 扩展。它与 Django 的机制完全类似。你可以在配置文件中配置 middleware 类。每个 middleware 可以处理请求和响应对象。
- views 模块的初始化处理。在 views 模块中，如果你写了一个名为 __begin__ 的函式，它将在执行要处理的 view 函式之前被处理，它相当于一个入口。因此你可以在这里面做一些模块级别的处理，比如检查用户的权限。因此建议你根据功能将 view 函式分到不同的模块中。

命令行工具

- 可以导出一个干净的工作环境。
- App 的创建，会自动包含必要的目录结构、文件和代码。
- 静态文件导出，可以将所有生效的 App 下的 static 导出到一个统一的目录。
- 启动开发服务器。

部署

- 支持 GAE 部署。
- 支持 Apache 下的 mod_wsgi 部署。

开发

- 提供开发服务器，并当代码修改时自动装载修改的模块。
- 提供 debug 功能，可以查看出错的代码，包括模板中的错误。

其他

- Uliweb 是一个 Demo 与源码合二为一的项目。它不仅包括所有核心代码，还同时包括了 uliwebproject（*http://uliwebproject.appspot.com*）网站的源码，同时还有其他的一些 Demo，所以你可以直接使用这些代码。
- 对于静态文件的支持可以处理 HTTP_IF_MODIFIED_SINCE 和 trunk 方式的静态文件处理。

简单入门

本教程将带你领略 Uliweb 的风采，这是一个非常简单的例子，你可以按照我的步骤来操作。我们将生成一个空的页面，它将显示“Hello, Uliweb”信息。

准备

从 *http://code.google.com/p/uliweb*（精巧地址：*http://bit.ly/OKSj3*）下载最新版本或从 svn 中下载最新版本，放在一个目录下。因为 Uliweb 本身包含 uliwebproject 网站的代码，所以在我们这个简单的例子中其实是不需要的，它们都存在于 apps 目录下。一种方式是你将它全部删除，因为我们会创建新的 app。另一种方式就是修改 apps 下的 settings.py 文件，只让我们新创建的 app 生效。再一种方法就是将代码导出到其他的工作目录下，这样环境比较干净，然后开始工作。这里我们将采用第三种方法。

创建新的项目

Uliweb 提供一个命令行工具 manage.py，它可以执行一些命令。在 Uliweb 的下载目录下，进入命令行，然后执行：

```
python manage.py export ../uliweb_work
```

这里 export 后面是一个目录，我是将它建在与 uliweb 同级的 uliweb_work 下了，你可以换成其他的目录。

在执行成功后（成功不会显示任何内容），从命令行进入新建的目录，在这个目录下是一个完整的 Uliweb 的复制，但是没有任何 APP 存在，所以是一个干净的环境。

创建 Hello 应用

然后让我们创建一个 Hello 的应用。在 uliweb_work 目录的命令行下执行：

```
python manage.py makeapp Hello
```

在执行成功后，你会在 apps 下找到：

```
apps/
  __init__.py
  settings.py
  Hello/
    __init__.py
    settings.py
    views.py
    static/
```

```
    templates/
```

好了，现在 Hello 已经创建好了。再一步就是如何创建“Hello, Uliweb”了。

输出“ Hello, Uliweb ”

打开 Hello/views.py，你会看到

```
#coding=utf-8
from uliweb.core.SimpleFrame import expose

@expose('/')
def index():
    return '<h1>Hello, Uliweb</h1>'
```

以上几行代码是在执行 makeapp 之后自动创建的。甚至我们都不用写一行代码，已经有一个 Hello, Uliweb 的 view 函式了。

@expose('/') 是用来处理 URL Mapping 的，它表示将/映射到它下面的 view 方法上。这样，当用户输入 *http://localhost:8000/* 时将访问 index() 方法。如果一个函式前没有使用 expose 修饰，它将不会与任何 URL 对应，因此可以认为是一个局部函式。

这里 index()没有任何参数。如果你在 expose 中定义了参数，它将与之对应。但因为这个例子没有定义参数，因此 index 不需要定义参数。

然后我们直接返回了一行 HTML 代码，它将直接输出到浏览器中。

启动

好了，让我们启动看一下结果吧。

在命令行下执行

```
python manage.py runserver
```

这样就启动了一个开发服务器。然后可以打开浏览器输入：*http://localhost:8000* 看到结果。

友邻篇

PCS400	GAE	388
PCS401	DHTML	395
PCS402	XML	402
PCS403	思维导图	411
PCS404	代码重构浅说	418

PCS400　GAE

Google 提供的免费 Web 应用空间！

概述



GAE（Google App. Engine）即 Google 应用引擎，是用于在 Google 服务器上创建和部署 Web 应用的免费开发平台，于 2008 年 4 月发布 beta 版本。App Engine 应用很容易被创建、维护和扩展，不需要服务器的维护，开发者只须上传自己的应用就可发布。

GAE 账号可以免费注册，也提供在 appspot.com 上的免费二级域名，也可通过 Google Apps 使用自己的域名。一个免费账号拥有 500MB 的持久存储，足够 CPU 配额和每月 500 万页访问的带宽。

特点

开发环境

GAE 可以很容易地建立一个 Web 应用程序，运行可靠。环境包括以下功能：

- 动态网页服务，完全支持通用网络技术；
- 持久存储，并提供查询、排序和交易功能；

- 自动扩展和负载平衡；
- 提供认证用户和使用 google 账户发送电子邮件的 API；
- 一个全功能的本地开发环境，在你的计算机上模拟 GAE 应用程序。

GAE 使用 Python 编程语言，其运行环境包含完整的 Python 语言和大多数的 Python 标准库。

沙箱

应用程序在一个安全，提供受限访问底层操作系统的环境中运行。这些限制允许应用程序引擎跨越多个服务器分发 Web 请求，启动和停止服务器，以满足交通的需求。沙箱将你的应用程序与安全可靠的运行环境，即独立于硬件，操作系统和物理位置的网络服务器隔离开。

受限安全沙箱环境包括：

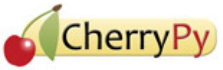
- 应用程序只能通过 URL fetch、邮件服务和 API 访问互联网上的其他电脑。 同时，其他电脑只能通过 HTTP（HTTPS）标准接口访问本应用程序。
- 应用程序无法写入文件系统。一个应用程序可以读取本地上传的文件。应用程序必须使用应用程序引擎数据存储接口访问所有数据。
- 应用程序在响应 Web 请求时运行相应代码，而且必须在几秒钟内返回响应数据。请求处理不能生成子进程或在响应发出之后执行代码。

纯 Python 运行环境

GAE 提供了一个使用的 Python 编程语言的运行环境。运行环境使用的 Python 版本是 2.5.2 ，包括了 Python 标准库。当然，调用库方法不能违反沙箱的限制条件。为了方便起见，几个核心功能不支持运行环境的标准库已被禁用。应用程序代码必须是 Python，C 扩展代码也是不支持的。

Python 环境提供了丰富的 API，如数据存储，谷歌账户，网址获取和电子邮件服务。应用程序引擎也提供了一个简单的 Python web 应用框架称为 Webapp，可以方便地开始创建应用。

GAE 还支持各种 Web 应用框架，只要框架支持 CGI/WSGI 接口。这其中包括：



CherryPy 框架，其官方网站为：*http://www.cherrypy.org/*

这是个古老的纯 Python Web 应用框架，关注对象化的发布管理，由于其稳定和简洁，有很多其他框架是直接在 CherryPy 的基础上再次开发而成的。



web.py 框架，其官方网站为：*http://webpy.org/*

这是极其简练高效的轻巧 Web 应用框架，是 *http://reddit.com/*（这是个非常流行的新闻推荐服务）的动力源泉。



PyLons 框架，其官方网站为：*http://pylonshq.com/*

这是关注高度灵活的“大框架”，PyLons 通过组合各种既有社区的作品，形成了一个高效灵活的应用框架，上述 *http://reddit.com/* 在后来就迁移到了 PyLons 框架中，而且开源了代码！下载入口：*http://code.reddit.com/*



Django Web 应用框架（基于版本 0.96.1 修订而成），其官方网站为：*http://www.djangoproject.com/*

Django 是最流行的 Python Web 应用框架之一，Python 的发明人 Giudo 也在公开场合多次表示喜欢 Django 框架；这是个 all in one 式的框架，支持快速架构起一个全功能的信息发布站点。不过 GAE 中的数据库和模板部分和 Django 中也有所不同。也可以在应用中使用上传第 3 方库，只要它们没有用到任何不支持的标准库模块。

数据存储

GAE 提供了一个强大的分布式数据存储服务，支持查询和事务。GAE 的数据存储不像传统的关系数据库，是由数据对象，或“实体”构成，他们具有特征。查询可以根据给定的条件特征检索实体或者按照属性值进行排序。属性值可以是任何支持的数据类型。

数据库 API 的数据模型接口可以定义数据实体的结构。这个数据模型可以表明某属性具有一个在某一特定范围内的值，如果没有给出值也提供一个默认值。 一个应用可以根据需求建立多个数据模型。

详细文档参考：The Datastore API - Google App Engine - Google Code

访问地址：*http://code.google.com/appengine/docs/datastore/*

精巧地址：*http://bit.ly/Wxy90*

实例

GAE 应用建立流程基本是这样的：

1. 获得 GAE 账号；
2. 下载开发包，当前（2008 十月中旬）最新版本是 1.1.5；

 下载地址：*http://code.google.com/appengine/downloads.html*

 精巧地址：*http://bit.ly/W06sN*
3. 在本地开发调试；
4. 通过开发包提供的 Python 脚本上传。

wekno

笔者拥有一个 GEA 账号，在此演示建立一个 GAE 中的“Hollo World”应用。

环境准备

下载 `google_appengine_1.1.5.zip` 开发包，解开压缩后在目录中建立 `wekno` 目录，目录名和在 GAE 中创建的应用名相同，创建相关脚本，与在目录中的组织类似。

```
google_appengine
  +-- ,,, 其他各个 SDK 中默认包含的目录
  +-- wekno          我的 GAE 应用目录
    |-- app.yaml     应用配置文件
    \-- hollo.py     默认页面脚本
```

开发部署

首先完成应用配置文件:app.yaml

```
application: wekno
version: 1
runtime: python
api_version: 1

handlers:
- url: /.*
  script: hollo.py
```

注意：

YAML（YAML——Ain't Markup Language，反标签语言的 YAML）和 GNU 一样，YAML 是一个递归着说“不”的名字。不同的是，GNU 对 UNIX 说不，YAML 说不的对象是 XML（官方网站为 *http://www.yaml.org/*），这是一种人性的、数据序列化的语言规约，有无数种开发语言的现实，在 GAE 中使用的是 PyYAML。

访问地址：*http://pyyaml.org/wiki/PyYAML*

精巧地址：*http://bit.ly/CuCpT*

GAE 选择 YAML 来组织配置文件，以便给将来支持其他语言的开发准备好平滑坚实的地基！

然后完成最简单的应用脚本：hollo.py

```
# -*- coding: utf-8 -*-
print 'Content-Type: text/plain'
print ''
print 'Hollo, World! this is WeKno!,,,'
print '是也乎?!'
```

先在本地运行测试一下，如图 PCS400-1 和图 PCS400-2 所示：

```
~/0penproj.s/GAE/google_appengine_1.1.2$ python dev_appserver.py wekno
INFO     2008-10-07 02:59:16,003 appcfg.py] Server: appengine.google.com
INFO     2008-10-07 02:59:16,012 appcfg.py] Checking for updates to the SDK.
WARNING  2008-10-07 02:59:16,718 datastore_file_stub.py] Could not read datastore data from /tmp/dev_appserver.datastor
WARNING  2008-10-07 02:59:16,719 datastore_file_stub.py] Could not read datastore data from /tmp/dev_appserver.datastor
ory
INFO     2008-10-07 02:59:16,751 dev_appserver_main.py] Running application wekno on port 8080: http://localhost:8080
INFO     2008-10-07 02:59:31,807 dev_appserver.py] "GET / HTTP/1.1" 200 -
```

图 PCS 400-1　在 SDK 中运行应用



图 PCS 400-2　在本地查看效果

然后上传到 GAE 空间，如图 PCS400-3 所示：

```
~/0penproj.s/GAE/sdk_google_appengine$ python appcfg.py update wekno
Loaded authentication cookies from /home/zoomq/.appcfg_cookies
Scanning files on local disk.
Initiating update.
Cloning 1 application file.
Uploading 1 files.
Closing update.
Uploading index definitions.
```

图 PCS 400-3　使用脚本上传应用

最后就可以在 GAE 网站中看到自个儿的应用了，如图 PCS400-4 所示：



图 PCS400-4 在 GAE 空间中查看效果

深入

毕竟到现在为止 GAE 还是 beta 版本，GAE 中的应用开发，和在普通的主机中自由组织自个儿的应用是不同的，有诸多限制，但是可以自在地使用各种 Google 不断公开的各种应用接口。

访问地址：*http://code.google.com/intl/zh-CN_ALL/more/*

精巧地址：*http://bit.ly/1RVuXm*

这是其他任何服务都无法相比的！

- 存储：实际在享用 GFS；
- 数据库：实际在享用 BigTable 对象数据库；
- 多媒体：实际可以使用 YouTube 数据仓库。

小结

Python 是市值已逾 1518 亿美元的 Google 公司使用的核心开发语言之一；GAE 是 Google 贡献给世界的免费应用空间服务，通过组合 Google 的多种服务，将 Google 宏伟的硬件平台，通过 Python 脚本，开发给普遍人一个轻便的操作渠道，可以帮助任何人，利用 Google 的计算平台发布自个儿的应用；两者相辅相成，直接激发了 Python 这一脚本语言在世界的知名度和发展（毕竟 Python 之父 Guido 老爹被 Google 挖来公司了）。所以，推荐读者也来体验一下 Google 的应用引擎服务，站在巨人的肩膀上自由发挥自个儿的想象力！

PCS401 DHTML

让网页活动起来!

概述

Dynamic HTML（动态 HTML）!

DHTML 是一种使 HTML 页面具有动态特性的艺术。

DHTML 是一种创建动态和交互 Web 站点的技术集。

对大多数人来说，DHTML 意味着 HTML、样式表和 JavaScript 的组合!

一个 DHTML 的结构大致可分为：

- 传统的 HTML。
- DOM（Document Object Model）文档对象模型，是 W3C 日前极力推广的 Web 技术标准之一，它将网页中的内容抽象成对象，每个对象拥有各自的属性（Properties）、方法（Method）和事件（Events），这些都可以通过 CSSL 来进行控制。IE 和 NS 的对象模型都以 W3C 的 DOM 为基准，加上自己的 Extended Object（扩展对象）来生成的。
- CSS（Cascading Styles Sheets）层叠式样式表。通过 CSS 样式表，可以自由定义各个网络组件的坐标位置（如静态、动态定位、或绝对坐标、相对坐标系统），及其他新增的标签、行为 （Behaviors）、图层（Layer）、字体和 Web 页格式等各种不同的设置。
- CSSL（Clent-Side Scripting Language）客户端脚本语言，主要有 JavaScript（JS），VBScript（VBS），JScript。Netscape 主要支持 JS，IE 主要支持 JS、VBS 和 JScript。

有了 Script，才可以让 CSS 设置为“行动”，从而达到整个服务器端与客户端的动态控制。

通过以上技术的综合，令普通的网页可以和访问者友好交互、活动起来。

HTML DOM

HTML Document Object Model（HTML 文档对象模型）：定义了访问和处理 HTML 文档的标准方法。

HTML DOM 把 HTML 文档呈现为带有元素、属性和文本的树结构（节点树）。

比如这一简单 HTML 页面，如图 PCS401-1 所示：

```
<html>
    <head>
        <title>
            文档标题
        </title>
    </head>
    <body>
        <h1>我的标题</h1>
        <a href="http://wiki.woodpecker.org.cn/moin/ObpLovelyPython">我的链接</a>
    </body>
</html>
```



图 PCS401-1　在游览器中看到的网页情景

DOM、HTML 文档中的每个成分都是一个节点。

DOM 是这样规定的：

- 整个文档是一个文档节点。
- 每个 HTML 标签是一个元素节点。
- 包含在 HTML 元素中的文本是文本节点。
- 每一个 HTML 属性是一个属性节点。
- 注释属于注释节点。

通过 DOM 就可访问 HTML 文档中的每个节点，如图 PCS401-2 所示。

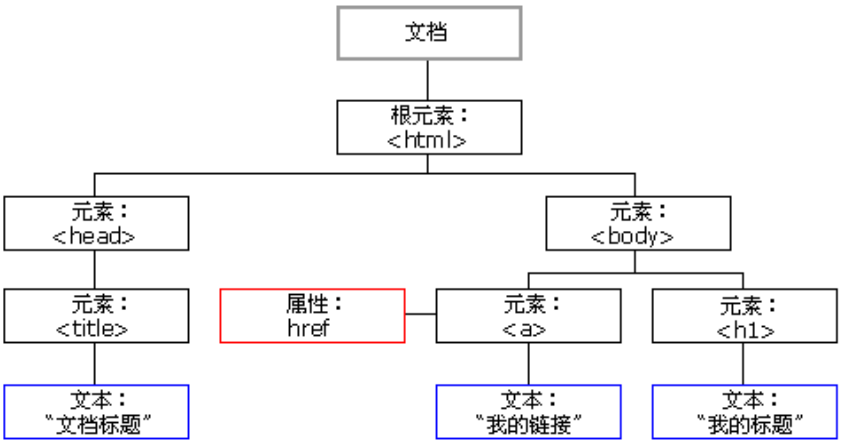


图 PCS401-2　DOM 理解后的节点树

可通过若干种方法来查找您希望操作的元素：

- 通过使用 getElementById() 和 getElementsByTagName() 方法。
- 通过使用一个元素节点的 parentNode、firstChild 以及 lastChild 属性。

HTML DOM 实例的访问地址：*http://www.w3school.com.cn/example/hdom_examples.asp*

精巧地址：*http://bit.ly/3coY8r*

DHTML CSS

CSS 是种网页表现形式的定义脚本语言，长得很像 Jscript。

比如说：

```
DIV#myfeel{
   visibility:hidden;
}
```

就是令所有用 层来包围的元素，暂时不显示！

通常，在 Web 页中指定 CSS 样式表的方式有 3 种：

1. 外部 CSS 样式表（External Styles Sheet）模式，它通过一个独立的 CSS 样式表文字文件（扩展名通常为 CSS）控制其他 Web 页。只要在需要指定样式的 Web 页中，设置一个链接至该 CSS 样式表文件，而且之后只要改变此 CSS 样式表内容，就可改变所有链接至该 CSS 样式表的 Web 页样式。
2. 内嵌 CSS 样式表（Embedded Styles Sheet）模式，它直接在 HTML＜body＞标签前设置一个样式标签，而这个设置会直接影响该 Web 页的样式设置。
3. 内部 CSS 样式表（Internal Styles Sheet）模式，它直接对 HTML 里的任何单一对象（如文字、图像等）进行样式设置，这种做法其实相当于利用文字的属性检查器。CSS 样式表直接定义的样式，只会影响单一选取的内容文字，而不会影响整个 Web 页的样式设置。

通常这 3 种方式浏览器的处理顺序为：内部 CSS 样式表模式、内嵌 CSS 样式表模式，外部 CSS 样式表模式，这也正是 CSS 的名称的由来，一层层地逐级解析和处理网页的表现。

DHTML 中的 CSS，利用 DOM 的约定，可以配合 CSSL 动态的对指定的页面元素进行外观的改变。

DHTML 实例的访问地址：*http://www.w3school.com.cn/example/dhtm_examples.asp*

精巧地址：*http://bit.ly/1PCYzL*

应用

当前 DHTML 也已经框架化，形成了 Ajax 技术群。

Ajax

Ajax（异步 JavaScript 和 XML）是 Jesse James Garrett 创造的一个术语，它是指一种基于标准的技术/设计模式，用来为服务器部署的应用程序开发比浏览器更好的用户体验。通过使用 Ajax，可以编写 JavaScript 代码来改进 HTML，创建出丰富的交互性用户体

验。例如，JavaScript 可以执行本地用户输入验证，为相同的数据提供不同的视图（条形图、表格、饼图等），或者通过浏览器的 XMLHTTPRequest 对象与应用程序的服务器组件进行异步的交互。

参考资料如下。

- Ajax 框架技术汇总

 访问地址：*http://ajaxpatterns.org/Ajax_Frameworks*

 精巧地址：*http://bit.ly/DRuCF*

- Ajax 框架汇总

 访问地址：*http://docs.huihoo.com/web/ajax/ajax-frameworks.html*

 精巧地址：*http://bit.ly/CHRHr*

- Ajax 框架汇总与对比

 访问地址：*http://www.444p.com/example/php-ajax/aid2639-ajax_frame/*

 精巧地址：*http://bit.ly/2C9hWG*

- Ajax 资源中心：Ajax 开发技术从基础入门到精通掌握所必需学习的文章，教程和参考资源

 访问地址：*http://www.ibm.com/developerworks/cn/ajax/*

 精巧地址：*http://bit.ly/lyptu*

简单地说，Ajax 技术，就是利用 XMLHTTPRequest 对象，通过 Jscript，在页面不刷新的情况下，以 JSON 数据包的格式，和服务器进行数据交互，并配合 CSS 在网页中表现出丰富动态效果的一组技巧；而 Ajax 框架,就是将丰富的 Ajax 技巧，打包起来，形成一组方便规范的调用接口，来帮助 Web 开发人员，快速完成基于 Ajax 技术的动态网页效果；然而各个 Ajax 框架出现的时机和背景思想各有不同，所以并不是所有 Ajax 框架对我们的应用情景都适用的，这里建议的选择原则，就是看相关社区是否活跃，相关第三方扩展开发是否丰富！

小结

DHTML 是种技术解决方案的组合，但是 DHTML 更像一种艺术行为，要求开发者使用有限的工具（HTML/CSS/JScrip）充分开动想象力，为自个儿的 Web 应用实现像桌面应用那样的操作和交互体验！

随着 Web2.0 的兴起，Web 应用要求单纯化、可扩展化、社会化，这其中 DHTML，或是进一步的 Ajax 的使用是至关重要的，因为它们统一了前后台数据交互的接口——JSON。



JSON（JavaScript Object Notation）是一种轻量级的数据交换格式。易于人们阅读和编写，同时也易于机器解析和生成。它是基于 JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999 的一个子集。

JSON 采用完全独立于语言的文本格式对其解析和生成，有各种语言的现实，Python 是内置支持的！

JSON 建构于两种结构：

- “名称/值”对的集合（A collection of name/value pairs）。不同的语言中，它被理解为对象（object）、纪录（record）、结构（struct）、字典（dictionary）、哈希表（hash table）、有键列表（keyed list），或者关联数组 （associative array）。
- 值的有序列表（An ordered list of values）。在大部分语言中，它被理解为数组（array）。

JSON 长得非常像 JScript：

```
{
    "glossary": {
        "title": "example glossary",
      "GlossDiv": {
            "title": "S",
           "GlossList": {
               "GlossEntry": {
                  "ID": "SGML",
                   "SortAs": "SGML",
                   "GlossTerm": "Standard Generalized Markup Language",
                   "Acronym": "SGML",
                   "Abbrev": "ISO 8879:1986",
                   "GlossDef": {
                     "para": "A meta-markup language, used to create markup
                      languages such as DocBook.",
                       "GlossSeeAlso": ["GML", "XML"]
                  },
                   "GlossSee": "markup"
```

```
            }
        }
    }
  }
}
```

对应 XML 的格式，表达相同的数据内容则是：

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
 <glossary><title>example glossary</title>
  <GlossDiv><title>S</title>
   <GlossList>
    <GlossEntry ID="SGML" SortAs="SGML">
     <GlossTerm>Standard Generalized Markup Language</GlossTerm>
     <Acronym>SGML</Acronym>
     <Abbrev>ISO 8879:1986</Abbrev>
     <GlossDef>
      <para>A meta-markup language, used to create markup
      languages such as DocBook.</para>
      <GlossSeeAlso OtherTerm="GML">
      <GlossSeeAlso OtherTerm="XML">
     </GlossDef>
     <GlossSee OtherTerm="markup">
    </GlossEntry>
   </GlossList>
  </GlossDiv>
 </glossary>
```

可以感觉出，使用 JSON 在 Ajax 应用中进行数据传输时，可以节省多少带宽了吧！

PCS402　XML

让电脑理解网页的技术

XML 的来源

XML 源自标准通用标记语言（Standard Generalized Markup Language，SGML），SGML 是一种通用的文档结构描述的符号化语言，主要是用来定义文献模型的逻辑和物理结构，但是由于其过于复杂和臃肿，只能够在大型企业和学术界使用，无法得到大规模应用。随后，于 20 世纪 90 年代提出的 HTML 技术得到了很大的发展，但由于 HTML 是面向表现的标记语言，而且结构固定，难以扩展，缺乏必要的语义信息，不适合用于信息交互。为了克服 HTML 和 SGML 的弊端，1996 年 W3C 专家组对 SGML 进行裁剪，形成 SGML 的精简子集，这就是现在我们所知的 XML。

XML 概念

XML 全称 EXtensible Markup Language，是一种用于描述数据文档中数据的组织和安排结构的一种规范。xml.org 通过以下陈述定义 XML：

- XML is a family of technologies / 一系列技术族。
- XML is a method for putting structured data in a text file / 一种用普通文本存放结构化数据的方法。
- XML is license-free, platform-independent and well-supported / 自由许可证，平台独立，支持性好。

- XML looks a bit like HTML, but isn't HTML / 看起来像 HTML，但不是 HTML。
- XML is new, but not that new / 新的，但不是完全新的东西。
- XML is verbose, but that is not a problem / 很详细，注重细节，但这不是问题。
- XML is text, but isn't meant to be read / 是文本，但并不意味着可读。

XML 也具有如下特性：

- Extensibility / 可扩展性：XML 让使用者根据需要自行定义标签。
- Structure / 结构化：XML 能够描述各种复杂的文档结构。
- Validation / 验证：XML 可以根据 DTD/XML Schema 对文件进行结构确认。

XML 和 HTML

如上所述，XML 看起来像 HTML，但决不是 HTML，通过下面这个例子可比较出两者的不同。

假如现在要表示出一 PC 的硬件配置信息，分别用 HTML 和 XML 表示为：

```
HTML 表示
<html>
  <body>
      <p>2200 MHz Pentium Iv
with 256K internal cache, 512K external cache, 256MB standard RAM, 1024MB
max. RAM</p>
     </body>
</html>
XML 表示
<pcinfo>
  <processor>
     <type>Pentium Iv</type>
     <speed>2200</speed>
     <intcache>256</intcache>
  </processor>
  <extcache>512</exctache>
  <ram>
     <standard>256</standard>
     <max>1024</max>
  </ram>
</pcinfo>
```

可以看到用 HTML 表示的那些数据仅仅是些字符串，没有什么语义信息，而 XML 除了具体有数据值还有标签，即该数据值表示的语义信息。进一步我们可以看出：**HTML 侧重于表示数据，即数据看起来像什么；XML 是描述数据，即数据是什么**。XML 的优势在于它对于同一内容可以有多种表示形式，内容也独立于应用，其数据可以像数据库一样查询和操作。注意：XML 的标签里包含的是纯信息，单独查看 XML 文件的内容并没有任何意义，需要应用程序根据特定需求读取，处理，显示，即要应用程序赋予它语义信息。

但是，**XML 并不取代 HTML**，因为它们的设计目标不同，XML 用于描述信息，而 HTML 用于显示信息。这是本质区别。

XML 语法介绍

下面的例子中描述了通信录信息的 XML 表示形式：

```
＜ ? xml version＝“1.0” encoding＝”GB2312” standalone＝”no”?＞
＜!DOCTYPE 联系人 SYSTEM”fclml.dtd”＞
＜?xml—stylesheet type＝”text / css” href＝“mystyle.css”?＞
   ＜联系人＞
       ＜姓名 曾用名=“张山”＞张三＜ / 姓名＞
       ＜用户号＞001＜ / 用户号＞
       ＜单位＞扬州大学＜ / 单位＞
       ＜EMAIL＞zhang@aaa. com＜ / EMAIL＞
       ＜电话＞ (0514)2345678 ＜ / 电话＞
       ＜地址＞
           ＜街道＞江阳中路 36 号＜ / 街道＞
           ＜城市＞扬州市＜ / 城市＞
           ＜省份＞江苏＜ / 省份＞
       ＜ / 地址＞
   ＜ / 联系人＞
```

可以看到，这个 XML 文档中依次有以下几个组成部分组成。

XML 声明

XML 文档以 XML 声明作为开始，它向解析器提供了关于文档的基本信息，必须出现在文件最前端。

```
<? xml version＝“1.0” encoding＝”GB2312” standalone＝”no”?>
      版本号          编码方式        是否为独立文档,值为 no 表示引用外部 DTD 文档
```

处理指令（PI）

处理指令（Process Instruction）是指示外部处理或应用程序的命令或宏。格式为：<?target instruction?>target 是一个在应用程序阅读 XML 时访问的外部应用程序，目的是执行某些计算；instruction 是指要提供给外部应用应用程序的命令。如前面的例子中，使用 PI 链接文档，组成样式表。

```
<?xml–stylesheet type=”text/css” href=“mystyle.css”?>
```

标签

XML 的标签是对文档内容进行描述的元数据，形式上是用一对尖括号括起来的，有开始标签（例如<姓名>）和结束标签（例如</姓名>），这些标签可以由用户自己创建。

元素

元素是开始标签、结束标签以及位于二者之间的所有内容。 如：<person>John</person>，其中“person”被称为元素名或标签名。须注意以下 4 点：

- 元素可以包含子元素，即可以嵌套。
- 元素区分大小写，如<Name>…</name>开始标签和结束标签是不匹配的。
- 元素不能交叉重叠，如 … 是不对的。
- 一些特殊字符不得出现在元素中，若一定需要表示特殊字符，可以使用转义序列，如下：

字符	名称	转义序列
>	Gt	>
<	Lt	<
“	Quot	"
‘	Apos	'
&	Amp	&

元素的命名要遵循以下规则：

- 元素名可以包含字母，数字和其他符号。
- 元素名不能用数字或标点符号开头。
- 元素名不能用 xml（或 XML，或 Xml 等）开头。
- 元素名不能包含空格。

XML 文档必须包含在一个单一元素中，这个单一元素称为根元素，它包含文档中所有文本和所有其他元素。在上面的示例中，XML 文档包含在一个单一元素＜联系人＞中。不包含单一根元素的文档不管该文档可能包含什么信息，XML 解析器都会拒绝它。

在 XML 文档中，是不能省去任何结束标签的。

如果一个元素根本不包含标签，则称为空元素；在 XML 文档的空元素中，可以把结束斜杠放在开始标签中。下面两个图像元素对于 XML 解析器来说是等价的：和

属性

它是一个元素的开始标记中的名称-值对。 如＜姓名 曾用名=“张山”＞张三＜ / 姓名＞，曾用名=“张山” 是 <姓名> 元素的属性。一个元素可以带任意数量的属性，且元素的属性不能嵌套，一个元素不能有两个相同属性名的属性，其属性值必须用引号括住。

注释

XML 的注释是以<!--开头 ，以-->结束，描述文档的功能,类似于程序设计语言的注释。但是它不能出现在前言之前，不能包在标签之中，但可以包住标签，也不能使用“--”字符串，更不能嵌套使用。

命名空间

XML 是一种用来定义我们自己的语言而定义数据的工具。在定义自己语言的过程中，总是可能面对在 Internet 上定义标签或元素陷入命名冲突。XML 命名空间（Namespaces）在 1999 年 1 月 14 日成为 W3C 的推荐标准，它的主要目的在于解决 XML 元素或属性名

称的冲突问题。XML 使用 URI（Uniform Resource Identifier）的引用作为 Namespace，只要将 URI 加到元素或属性名称的前面，就能使名称具有唯一性。如：

```
<description>
   <{http://www.w3.org/TR/xhtml1}head>
      <{http://www.w3.org/TR/xhtml1}title>Book
      </ {http://www.w3.org/TR/xhtml1}title>
   </{http://www.w3.org/TR/xhtml1}head>
</description>
```

可以写成：

```
<description xmlns:ht=" http://www.w3.org/TR/xhtml1" >
     <ht:head>
     <ht:title>Book</ht:title>
     </ht:head>
</description>
```

命名空间的声明由 xmlns:prefix 组成。声明固定为 xmlns，之后用':'隔开，prefix 不能包含 xml 的字样。名称空间定义中的字符串仅仅是字符串，上例中，也可以定义 xmlns:addr="yzu"，这也是有效的。默认的命名空间采用 xmlns="someuri"来表示没有 prefix 的元素使用此命名空间。

```
<library xmlns="http://www.w3c.org">
     <book>xml bible</book>
     <author name="abcd">……</author>
</library>
```

如果没有默认的命名空间，没有 prefix 的元素不属于任何命名空间。命名空间的作用域仅限于命名空间声明的元素以及子孙元素。

```
  <library>
     <bk:book xmlns:bk="http://www.w3.org">
         <bk:title>…</bk:title>
   </bk:book>
   <bk:description>…</bk:description>    # 这个是错误的，因为 bk 已经超出作用域了
</library>
```

DTD 和 XML Schema

文档类型定义（Document Type Definition），简称 DTD。DTD 可以定义在普通 XML 文档中出现的元素、这些元素出现的次序、它们可以如何相互嵌套以及 XML 文档结构的其它详细信息。DTD 是最初的 XML 规范的一部分，但是 DTD 语法不同于 XML 语法。DTD

的一般写法是：<!DOCTYPE 根元素名[…[内容声明]…]>。元素和属性是两个主要声明：

```
<! ELEMENT element_name content_spec >
          元素名称       元素内容的类型
<! ATTLIST element_name attribute_name type default >
        所属元素的名称     属性的名称     类型   指默认值
```

DTD 的局限有以下几点：

- DTD 本身不是 XML 格式，无法使用标准的编程方式进行 DTD 维护。
- DTD 不支持命名空间。
- DTD 仅支持有限的数据类型，在大多数应用环境下表示能力不足。
- 无法重复使用先前声明的元素或属性。
- 不易描述元素出现次数。

由于 DTD 的局限性，进一步提出了 XML Schema。XML Schema 是 W3C 的推荐标准，于 2001 年 5 月正式发布，经过数年的大规模讨论和开发，最终成为全球公认的 XML 环境下的首选建模工具，已经基本取代了 DTD 的地位。XML Schema 正式定义 XML 实例文件（instance document）的结构，即验证 XML 实例文件是否合法；也定义了出现在 XML 文件中的元素和属性的数据类型及结构。XML Schema 预定义的元素和属性在名称空间（*http://www.w3.org/2001/XMLSchema* 精巧地址：*http://bit.ly/1cK2ff*）中定义，主要包括以下一些基本元素。

- Schema：XML Schema 的根元素。
- SimpleType：定义数据类型。
- ComplexType:声明 XML 元素的结构。
- Element：声明 XML 子元素。
- Attribute：声明 XML 元素属性。

XML Schema 中定义的数据类型称为简单类型，它很好地规范了 XML 文档的文本内容的形式和语义。而且因为 XML Schema 本身是 XML 文档，其定义的数据类型可以直接在 XML 文档中使用。XML Schema 中的简单类型可以分为三种：原子类型、列表类型和联合类型。

XML Schema 内置了 40 多种简单类型，常见的有：string（字符串）、Boolean（布尔类型）、URI reference（URI 引用）、float（浮点数字）、double（双精度浮点数）、ID（身份号）、decimal（十进制数字）、 ENTITY（实体）和 TimeDutation（时间数据）等。

工具和资源

XML 开发环境

- Altova XMLSPY（商业软件、可视化，支持拖曳）

 访问地址：*http://www.altova.com/download.html*

 精巧地址：*http://bit.ly/JlXgU*

- Liquid XML Studio 2008（自由软件）

 访问地址：*http://www.liquid-technologies.com/Download.aspx*

 精巧地址：*http://bit.ly/dQqia*

XML 文档

- W3C 官方 XML 教程

 访问地址：*http://www.w3.org/TR/xml/*

 精巧地址：*http://bit.ly/hzdab*

- XML Schema：提供大量 XML 相关的编辑、解析工具

 访问地址：*http://www.w3.org/XML/Schema*

 精巧地址：*http://bit.ly/2PssD4*

- 中国万维网联盟（W3CHINA）：关于 XML 的专业性中文技术站点

 访问地址：*http://www.xml.org.cn/*

 精巧地址：*http://bit.ly/1CPto0*

- XML 问题专栏

 访问地址：*http://www.ibm.com/developerworks/cn/xml/x-matters/*

 精巧地址：*http://bit.ly/47Z8w1*

小结

IBM 从 20 世纪 60 年代就开始发展的 GML，到 1986 年标准化成 SGML（ISO 8879），虽然好用，但是太复杂。随着互联网的发展，在 1989 由 CERN（Conseil Europeen pour la Recherche Nucleaire）简化成 HTML，随着 Internet 的发展，HTML 快速发展到 4.0 版，但是有失控的趋势，而且面对爆炸式的网页增长，根本没有什么好办法来让电脑理解网页内容，从而更加聪明地帮助人们寻找可用的资料。所以 1996 年合理简化 SGML 提出 XML。1998 年，成为 W3C 标准（XML1.0）。但是，人们发现 XML 配套的技术和软件平台太少，网络中不断增长的还是对于电脑来讲一片混沌的 HTML 页面，于是在 2000 年底，XML 向 HTML 妥协，基于 HTML4.0.1 发布了 XHMTL1.0，将 HTML 改进成吻合 XML 规范的格式，从而部分解决了电脑理解网页的问题。这个围绕着网页的故事还在继续，而 XML 已经跨出了网页的领域，在数据挖掘、分布式计算、语义网研究等方面不断地快速成长着。就行者来看，XML 可以简单理解成:

1. 结构化数据的容器标准；
2. 数据处理的结构化标准。

再简单说，XML 就是当前最方便和通用的数据交换格式！凡是程序间/系统间/网站间的数据交互，优先考虑使用 XML 准没错!

PCS403　思维导图

千言万语不及一张图

概述

“使用思维导图是波音公司的质量提高项目的有效组成部分之一。这帮助我们公司节省了一千万美元!”

——Mike Stanley，波音公司

美国波音公司在设计波音 747 飞机的时候就使用了思维导图。据波音公司的人讲，如果使用普通的方法，设计波音 747 这样一个大型的项目要花费 6 年的时间。但是，通过使用思维导图,他们的工程师只使用了 6 个月的时间就完成了波音 747 的设计! 并节省了一千万美元。 思维导图的威力惊人吧?!

“我们的课程建立在思维导图的基础上。这帮助我们获得了有史以来最高的毕业分数。思维导图教学必然是未来的教学工具。”

——Jean Luc Kastner，高级经理，惠普公司

什么是思维导图？为什么思维图谱这么神奇？

人的记忆能力实质上就是向大脑储存信息，以及进行反馈的能力。人的大脑主要有神经细胞构成，每个神经细胞的边缘又都有若干向外突出的部分，被称作树突和轴突。在轴突的末端有个膨大的突起，叫做突触小体。每个神经元的突触小体跟另一个神经元的树突或轴突接触。 这种结构叫做“突触”。神经元通过“突触”跟其他神经元发生联系，并且接受许许多多其它的神经元的信息。神经元传递和接受信息的功能，正是大脑具有记忆的生理基础。每个神经元上有多少个突触呢？有人估计，在人们大脑皮层每个神经

元上平均有三万个突触。那么，人脑有多少神经元呢？大约有 140 亿个。这 140 亿个神经细胞之间的突触联系的，用天文数字也难以表达。这样的结构特点，就使大脑成为一个庞大的信息储存库。一个人脑的网络系统远比当今英特网还复杂。科学家认为，一个人大脑储存信息的容量，相当于十亿册书的内容，一个人的大脑即使每一秒钟输入十个信息，这样持续一辈子，也还有余地容纳别的信息。 这说明：我们大脑的记忆容量是无限的，有很大的记忆能力，而且我们的记忆天然是立体网状的，不是日常各种记录媒体那样平面的线性的！

爱因斯坦曾这样描述他的思考问题时的情景：“我思考问题时，不是用语言进行思考，而是用活动的跳跃的形象进行思考，当这种思考完成以后，我要花很大力气把他们转化成语言。”显然，正是左右脑协同工作，使人类具有感知力、创造力。

思维导图是表达发射性思维的有效的图形思维工具。思维导图运用图文并重的技巧，把各级主题的关系用相互隶属与相关的层级图表现出来，把主题关键词与图像、颜色等建立记忆链接，思维导图充分运用左右脑的机能，利用记忆、阅读、思维的规律，协助人们在科学与艺术、逻辑与想象之间平衡发展。

“思维导图”是图形化的对思维过程的导向和记录。思维导图促进思维的发现，并能记录这个发散过程。

使用思维导图的好处：

1. 使用大脑所有皮层技巧，利用了色彩、线条、关键词、图像等，因此可以大大加强回忆的可能性。
2. 通过动手绘制思维导图可以激发大脑的各个层次，使大脑处于警醒状态，在记忆的时候更加有技巧。
3. 整个思维导图就是一个大的图像，是整个信息资讯的整体架构，使大脑希望回到它们中间去，因而又一次激发自发回忆的可能性。
4. 它们的设计极为简单，而且结构清晰，层次分明，便于对信息的组织和管理，因而可以帮助记忆。
5. 使用助记的思维导图会激发大脑准备好记忆，因而，每用一次，大脑基本的记忆技巧水平就会提高一次。
6. 反映了人们创造力思维过程，使得我们的思维过程可视化和可操作化，因此也就同时加强了创造性思维技巧。
7. 因为在制作思维导图时，要求我们把关键性的内容记下来，因此可以促使我们在学习和倾听的阶段都保持着较高水平的回忆。

8. 关键词和图像的应用，促使人们使用个人所有的联想能力，思维导图本身的形状加强了大脑物理印迹和网络开发能力，因此就增加了回忆的可能性。
9. 因为使用了左脑和右脑的所有技巧，提供了一个“十拿九稳”的记忆方法，增大了个人的信心，动机和普遍的心理作用。

在 CDay–1 日的故事中，笔者就针对中文问题的解决，记录成了一个思维导图，如图 PCS403-1 所示：

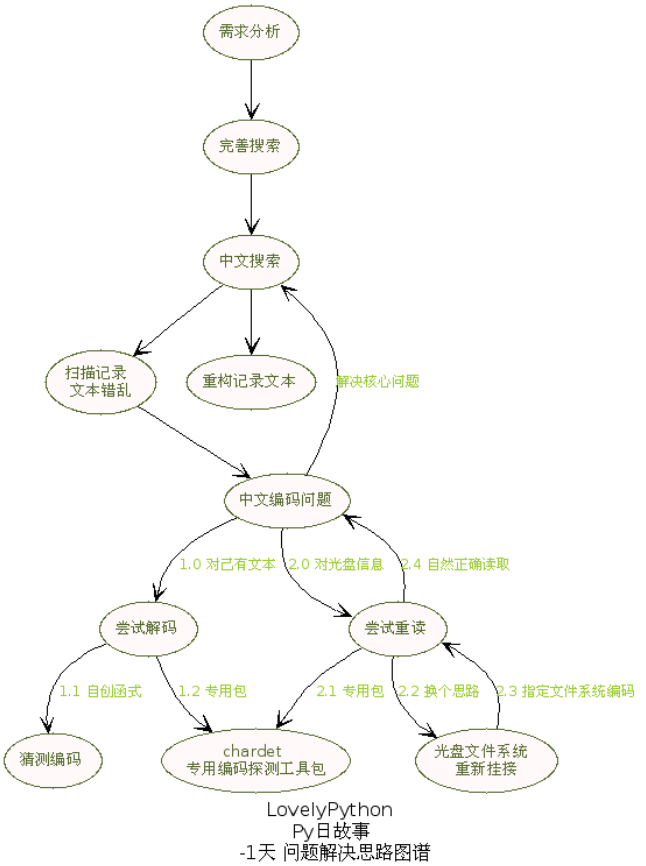


图 PCS403-1

应用

思维导图的使用，可以直接利用纸和笔，如图 PCS403-2 所示。



图 PCS403-2

（此图摘取自[原创]泡泡茶的《启动记忆》《掌握记忆》章节总结 [栖息谷·管理人网络社区]，网站地址：*http://bbs.21manager.com/*）

但是一般人的手绘能力并不强，只是使用文字来进行导图记要，如图 PCS403-3 所示。

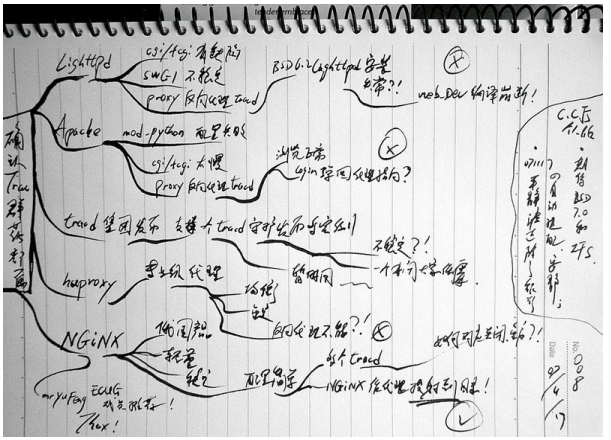


图 PCS403-3

（此图摘取自 *http://zoomquiet.blogspot.com/2007/11/grblogging-071117.html*，精巧地址：*http://bit.ly/2adpVo*）

而且，在网络中分享传播手绘的导图比较麻烦。所以，这里推荐一个轻巧实用的思维导图工具——FreeMind。

Freemind 是一款简单易用的开源思维图谱（MindMap）软件，或称树编辑器。通过它可以创建可折叠的树形结构，节点可以是纯文本，也可以添加颜色、图标，节点也可以具

有云的形状及其他的图形。方便的折叠功能和强大的搜索功能使 FreeMind 成为一款颇有价值的知识库工具，同时也能够通过键盘进行控制。

使用 FreeMind

FreeMind 可以应用于以下方面：

- 跟踪项目，包括子任务，子任务的状态以及时间纪录；
- 项目工作区，包括必要的链接，如文件，可知性文件，信息来源及信息；
- 互联网搜索工作区，使用 Google 及其它资源；
- 保留中小尺寸的备忘录，包括一些链接以扩展内容。这些备忘录也可以称作知识库；
- 散文写作及头脑风暴，使用颜色区分哪些散文是打开的，哪些已完成，哪些未开始。节点可以显示散文的大小；
- 保存小型的结构化的数据库，可以动态改变，非常灵活。虽然查询功能有限，但是可以根据你的喜好，保存任何信息，如联系人、名片、医疗记录等；
- 网络收藏夹或书签，可以任意设置颜色和字体。

FreeMind 特性

从早期版本开始，FreeMind 就支持折叠功能，这是它的核心特性，它的特性还包括：完全支持节点中 HTML 链接，链接可以是网站地址或者本地文件；快速的点击导航，包括折叠/打开，链接点击导航。拖动图谱的背景就可移动图谱，也可使用鼠标滚轮移动；它具有取消操作功能；还拥有即拖即放功能，包括节点拷贝或节电样式拷贝；多个节点的拖放功能；从外部拖放文本或文件列表；拷贝和粘贴功能，粘贴 HTML 中的链接和文件列表，或者拷贝纯文本和 RTF（MS Wordpad、MS Word、MS Outlook 消息）；查找功能，通过“find next”一个接一个显示查找到的纪录；它能够使用和编辑多行节点；能够使用内置的图标，颜色，不同的字体装饰节点，而且可以将图谱输出成 HTML 格式，同时还可以低成本低风险地将图谱转移到其他的思维图谱工具，因为 FreeMind 通过 XML 格式存储信息。

最大的特点是使用方便，能够以 xml 的格式输出。推荐下载使用 0.8rc2 的版本，它使用非常流畅。另外，FreeMind 使用 Java 编写，支持使用 Jython 编写扩展插件。由于文件格式是 xml，因此便于扩展，比如：存放更多的扩展属性信息。

FreeMind 实例

Google 的发展思路如图 PCS403-4 所示。

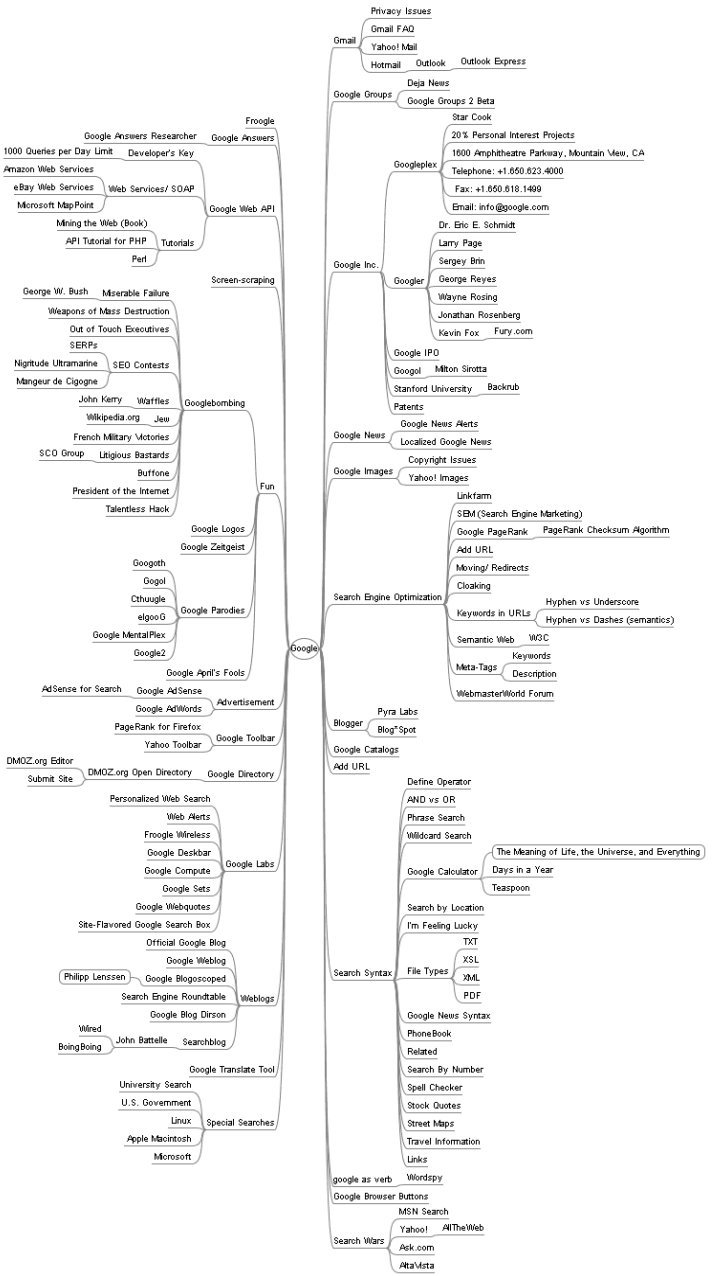


图 PCS403-4

知识点图谱

- Vi 快速学习图谱

 访问地址：*http://wiki.woodpecker.org.cn/moin/ViQuickIn*

 精巧地址：*http://bit.ly/2uYdb4*

- MoinMoin 学习图谱

 访问地址：*http://wiki.woodpecker.org.cn/moin/MoinMoinTutMap*

 精巧地址：*http://bit.ly/4k3CwO*

- “提问的智慧”图谱

 访问地址：*http://wiki.woodpecker.org.cn/moin/AskForHelp*

 精巧地址：*http://bit.ly/2JGVxM*

- 思维图谱链接整理

 访问地址：*http://wiki.woodpecker.org.cn/moin/MindMapping*

 精巧地址：*http://bit.ly/1pbStr*

小结

借助工具，我们可以利用思维导图来记录思维过程，协助厘清概念关系，展示设计思想，浓缩复杂知识体系以便加速记忆。思维导图可以用来作什么并没有强行约定，但是图形化的思维导图已被证明是极其有效地提高沟通和增强记忆的好工具！大家得空应该体验一下。

PCS404　代码重构浅说

重构是必要的浪费

概述

Refactoring：修改内部结构（设计）而不影响外部行为。

代码和软件的重构对于用户是没有意义的，所以是种浪费。但是，对于程序员来说，代码和软件的重构是绝对必要的！因为它可以：

- 加深对程序的理解。
- 使程序更加容易被理解。
- 找到隐藏的 BUG。
- 提高开发速度。
- 不知不觉中完成对软件的设计增进。
- 神奇地提高代码的可维护性/复用性/健壮性……

应用

首先推荐一本专门的好书：《重构——改善既有代码的设计》，封面如图 PCS404-1 所示。

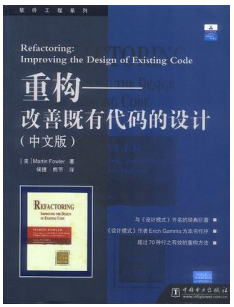


图 PCS404-1

豆瓣介绍：*http://www.douban.com/subject/1229923/*

精巧地址：*http://bit.ly/2mvGyj*

其实在前面的 CDay/KDay 故事中，针对同一功能的实现，先后使用不同的思路和代码在保持对外接口不变的情况下，反复使用更精简、明了、易读的代码重新实现，这本身就是重构!

```
1 import os
2 export = ""
3 for root, dirs, files in os.walk('/media/cdrom0'):
4   export+="\n %s;%s;%s" % (root,dirs,files)
5 open('mycd2.cdc', 'w').write(export)
```

演变成：

```
1 import os
2 export = []
3 for root, dirs, files in os.walk('/media/cdrom0'):
4    export.append("\n %s;%s;%s" % (root,dirs,files))
5 open('mycd2.cdc', 'w').write(''.join(export))
```

又变成：

```
1 import os
2 def cdWalker(cdrom,cdcfile):
3    export = ""
4    for root, dirs, files in os.walk(cdrom):
5       export+="\n %s;%s;%s" % (root,dirs,files)
6    open(cdcfile, 'w').write(export)
7 cdWalker('/media/cdrom0','cd1.cdc')
```

就是根据经验或是直觉，将具有“坏味道”的代码不断改进为更舒服的代码。

注意：所谓代码的“坏味道”，就是程序员们在多年各种开发活动中总结出来的，令代码的扩展和维护趋向混乱和不可控的态势，具体请参考《重构——改善既有代码的设计》一书中详细介绍的几十种“坏味道”及其应对方法。

问题

什么时候应该进行重构？这个问题随着我们的经验积累，答案会增殖的！所以，可以从反向进行解答“**什么时候不应该进行重构**”？

1. 现有的程序无法运行，此时应该是重写程序，而不是重构。
2. 程序到了最后的交付期限。

重构活动中的难题，当前可以确认的，无法轻快地当场进行重构的难题有以下几类：

- 关系数据库与面向对象编程的问题:在对象模型和数据库模型之间插入一个分隔层，这就可以隔离两个模型各自的变化。升级某一模型时无须同时升级上述的分隔层。这样的分隔层会增加系统复杂度.但是能增加灵活度。
- 修改接口的问题:修改已发布的接口，因为已发布的接口会供外部人员（其他公司）使用，因此，修改接口会导致引用接口的其他程序不修改程序就无法运行。修改接口的最好的办法是增加一个新的接口，让旧接口调用新接口。这样原来的程序就不用修改了。对于接口的另一个建议是尽量不要发布接口。

探讨

由于重构法则在长期探索中形成了各种模式，进而已经被包含在一些高级 IDE 环境中!

对于 Python，Bicycle Repair Man 可以自动地对指定代码进行指定模式的自动化重构。

官方网站：*http://bicyclerepair.sourceforge.net/*

精巧地址：*http://bit.ly/fuAGL*

进一步请参考维基百科“软件重构”。

访问地址：*http://zh.wikipedia.org/wiki/软件重构*

精巧地址：*http://bit.ly/1CRg9I*

小结

重构是最简单易行的提升代码品质的手段，不过，其实施和管理方式是属于 XP～极限编程思想，不是我们日常学校里念叨的“瀑布式开发流程”，重构要求的是就地、随时进行的勇气、技巧及热情!只要我们非常看重我们自个儿写出来的代码，期望这些代码可以长久地反复地，稳健地运行/使用/分享下去，那么就开始重构吧!